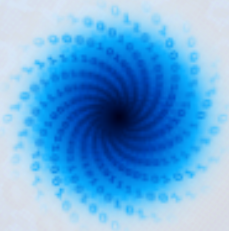
MAchinE Learning for Scalable meTeoROlogy and cliMate

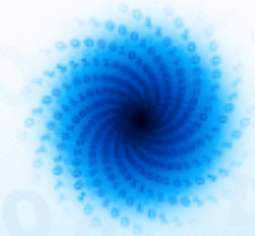


MAELSTROM

# Mid term hardware performance benchmarking

E4 Computer Engineering

www.maelstrom-eurohpc.eu




# D3.6 Report on hardware performance benchmarking for ML solutions from D1.3 on a number of hardware configurations

**Author(s):** E4 Computer Engineering

Mattia Paladino (E4), Daniele Gregori (E4)

**Dissemination Level:** Public

**Date:** May 12, 2023
**Version:** 1.0
**Contractual Delivery Date:** 05/2023
**Work Package/ Task:** WP3/ T3.3
**Document Owner:** E4 Computer Engineering
**Contributors:** FZJ

**Status:** Final

# MAELSTROM

## Machine Learning for Scalable Meteorology and Climate

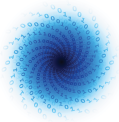
**Research and Innovation Action (RIA)**
**H2020-JTI-EuroHPC-2019-1: Towards Extreme Scale Technologies and Applications**

**Project Coordinator:** Dr. Peter Dueben (ECMWF)
**Project Start Date:** 01/04/2021
**Project Duration:** 36 months

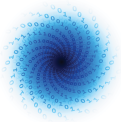
**Published by the MAELSTROM Consortium**

**Contact:**
ECMWF, Shinfield Park, Reading, RG2 9AX, United Kingdom
Peter.Dueben@ecmwf.int

The MAELSTROM project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955513. The JU receives support from the European Union's Horizon 2020 research and innovation programme and United Kingdom, Germany, Italy, Luxembourg, Switzerland, Norway.

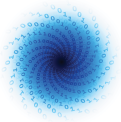


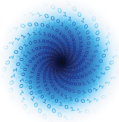
# Contents

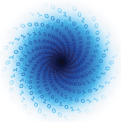




# List of Figures

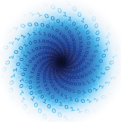


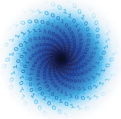






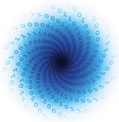
# List of Tables





# 1 Executive Summary

Based on the results of the early benchmarking described in D3.4, a second phase of benchmarking was conducted to investigate the applications performance further. This second phase of benchmarking aimed to identify areas for improvement and optimization of the application's utilization of the available hardware, taking into account the performance metrics established in the early benchmarking. In the second phase of benchmarking, we have expanded our evaluation platform to include a wider range of hardware configurations, allowing us to assess how the MAELSTROM applications performed on a more diverse set of systems. By analyzing the application's performance on these different configurations, we were able to identify specific hardware setups that were particularly effective in optimizing performance and energy efficiency. Furthermore, by comparing the results of the second phase of benchmarking to the earlier benchmarks, we were able to assess the effectiveness of the optimizations made to both the applications and hardware systems. This allowed us to identify areas for further improvement and to make informed decisions about the hardware systems that should be used for future benchmarks.



# 2 Introduction

## 2.1 About MAELSTROM

MAELSTROM aims to create Europe's next-generation computer architecture by co-designing custom compute system designs for optimal application performance and energy efficiency, along with a software framework to improve usability and training efficiency for large-scale machine learning applications in weather and climate science.

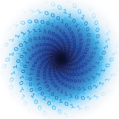
To achieve this, MAELSTROM will benchmark these applications across various computing systems based on energy consumption, time-to-solution, numerical precision, and solution accuracy. Customised compute systems will be designed that are optimised for application needs in order to enhance Europe's high-performance computing portfolio and to pull recent hardware developments towards the unique requirements of weather and climate applications. The MAELSTROM software framework will enable scientists to apply and compare machine learning tools and libraries across a wide range of computer systems with ease. This will be supported by a user interface that links application developers with compute system designers. Also, during the development phase, automated benchmarking and error detection of machine learning solutions will be conducted. These tools will be published as open source.

The MAELSTROM machine learning applications will cover all the key components involved in the workflow of weather and climate predictions. This includes processing of observations, assimilation of observations to generate initial and reference conditions, model simulations, as well as post-processing of model data and development of forecast products. For each application, benchmark datasets with up to 10 terabytes of data will be available online for training and machine learning tool-development on the fastest supercomputers in the world. The machine learning solutions developed by MAELSTROM will serve as a blueprint for future machine learning applications on supercomputers.

## 2.2 Scope of this deliverable

### 2.2.1 Objectives of this deliverable

Deliverable 3.6 is a report on the work done for Task 3.3, as mid term benchmarking of ML solutions depicted in D1.3 on a wider range of hardware and monitoring tools



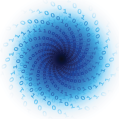
to investigate new configuration compared to D3.4. New monitoring tools include intelligent Poweer Distribution Unit (PDUs) adopted in E4 premesis to measure the overall power of each single server. Jube monitoring software developed at JSC to collect information from OS and applications sensors (i.e. GPU power, job execution time, data load time).
Deliverable 3.6 is the second MAELSTROM deliverables that provide the basis to benchmark the applications on HPC hardware. It provides a large variety of metrics and plot related to application executions on heterogeneous HPC systems to allow performance evaluation.

### 2.2.2 Work performed in this deliverable

The performance evaluation metrics were agreed upon with the WP1 application developers and are the same as those used in the previous benchmarking phase in D3.4 with the addition of some new metrics related to the energy consumption of each node. A spreadsheet was provided to the application developers to enter their benchmark results on the available HPC machines.
In the second benchmarking phase, application developers were granted access to resources at JSC and E4. Information on system access, benchmark runs, and metric measurement was compiled and documented on the project's Confluence page, where it was accessible to all project members.
The application developers ran the benchmarks, and the results were recorded in the spreadsheet.
The results were analyzed to examine performance, scalability behavior, energy efficiency, and potential issues. In this phase, we used multiple evaluation platforms with different configurations to ensure a thorough analysis of the applications' performance.

### 2.2.3 Computing configuration and Storage

The computational systems used are the same as those described in deliverable D3.3. ARM architectures and modifications to previous configurations, in terms of RAM, A100x accelerators, and FPGAs were proposed to the developer community. However, these alternatives have not been utilised yet as additional work on the instalation of machine learning libraries and the porting of the applications is required.
From the previous experience gained in D3.4, special attention was paid in this deliverable to the use of data storage systems to understand which ones improve performance. Two types of data areas were available at E4: a common area ex-



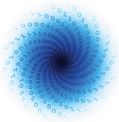
ported among all nodes via NFS protocol with a 1 Gb/s bandwidth limit, and a local disk in NVMe technology with single-stream sequential write and read performance of 2.5 GB/s and 1 GB/s. Although the NFS filesystem is limited by Ethernet network bandwidth it has enabled caching. After the first read of the dataset, the data resides on the local memory of the compute server improving the load time of subsequent training.

Two types of parallel filesystems were provided at the JSC center, both based on GPFS, whose performance is described in this paper[1]. The High-Performance Storage Tier (HPST) is a specialized low-capacity tier that uses non-volatile memory technologies (NVMe) to optimize performance. Acting as a cache layer on top of the SCRATCH file system, the HPST provides high bandwidth and low latency access for I/O in compute jobs. The HPST is based on the "Infinite Memory Engine (IME)" architecture, developed by DataDirect Networks (DDN). Using a client-server model, the server creates a cache on top of the SCRATCH file system, mounted on all servers, and utilizes the 10 NVMe disks on each server for cache storage. With up to 2 TB/s bandwidth for optimized parallel I/O, a single client node can achieve up to 8 GB/s. Data can be staged in and out of the underlying SCRATCH file system using special commands. Data allocated or prestaged on the HPST can be accessed using POSIX, MPI-IO (when compiled using ParaStationMPI, otherwise it defaults to POSIX) or the IME native interface. Our initial experience shows that both MPI-IO and IME native can result in better performance, but do require some recompiling or porting efforts. Additionally, we have identified that single process access with specific patterns to the SCRATCH file system can be faster than POSIX access to HPST. This can be explained by the IME client’s use of FUSE in Direct-IO mode limiting file system Readahead and caching.

### 2.2.4 Deviations and counter measures

The deliverable was delayed by 6 weeks due to the need for additional time to collect and analyze the data.

[1] https://jlsrf.org/index.php/lsf/article/view/180/pdf

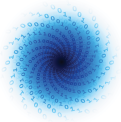


# 3 Metrics

The metrics selected for performance evaluation fall into four categories: time-related metrics, model-related metrics, energy-related metrics and general score metric. The following metrics have been measured and documented for all of the applications:

- Time-related
  - Total runtime
  - Total training time
  - Loading Data Time
  - Min. training time per epoch
  - Max. training time per epoch
  - Avg. training time per epoch
  - First epoch training time
  - Avg. training time per iteration
  - Saving model time
- Model-related
  - Final training loss
  - Final validation loss
- Energy-related
  - Max. GPU power
  - GPU energy consumption
  - Total node energy consumption
- General Score
  - Action

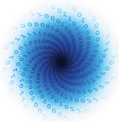
From a general benchmarking perspective, metrics such as total runtime, training time, and data loading and storing times are relevant. Timing metrics provided by the ML frameworks, such as epoch training time, are also included. Additionally, the final training and validation loss metrics are important from the ML perspective. In order to measure energy efficiency, we have recorded the power and energy consumption of the GPU, as well as the energy consumption of the nodes used in the



benchmarks. Moreover, we defined a general score metric named "Action", the name is due to the metric unit energy × time, that shows the server quality related to the application, the value that minimize the action represents the best configuration that optimize the application execution performance.

In E4 premises, the node power consumption was recorded for all benchmarking runs in this phase thanks to the presence of an intelligent Power Distribution Unit (PDU), which allows the overall power measurement of the single server node and enables the automatic recording of power consumption data at regular intervals during the benchmarking runs. The GPU power consumption could be obtained via commands `nvidia-smi` for NVIDIA GPU and `rocm-smi` for AMD GPU. However, single GPU power consumption is not available because the required Jube interface hasn't been configured yet.
In Julich premesis the infrastructure for measuring GPU and full-node energy consumption is currently under development. However some tools are already available. On JSC systems, GPU power consumption is provided by the LLview[2] job report - internally, `nvidia-smi` is used to measure GPU power periodically during the job execution. The recorded power consumption data was then used to calculate the energy efficiency metrics for each benchmarking run.

[2] `https://www.fz-juelich.de/ias/jsc/EN/Expertise/Support/Software/LLview/_node.html`



# 4 Benchmarks

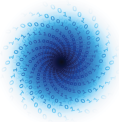
Applications were mainly benchmarked on 4 different systems: JUWELS Booster, JUWELS Cluster, E4's Intel Cluster, and E4's AMD Cluster. The focus was both on training benchmarks and inference performance. For some applications, multiple configurations were investigated. In cases where inconsistencies were found in the metrics of the first 3 runs, the developers were asked to perform more measurements.
In addition to the metrics mentioned in Section 3, job-specific information was recorded for each job, which enables querying of job-specific information at a later stage. The following details were recorded:

- Number of CPUs used
- Number of GPUs used
- Number of Nodes used
- Number of MPI tasks
- Job ID
- Node IDs

For each of the applications an overview of the application is given, including the following characteristics of the application:

- Memory training dataset
- Memory validation dataset
- Training samples
- Input shape sample
- batch size
- Trainable parameters
- Non-trainable parameters
- Loss function
- Experimental notes

In the following sections, we provide new interpretations of the benchmark data, while the total raw data is available in the appendix.



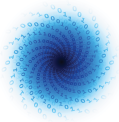
## 4.1 AP 1

### 4.1.1 Notes

Deliverables D1-3 and D3-4 revealed that data loading was a major bottleneck for AP1. Since then, the data loader has been rewritten to improve performance. Earlier implementations achieved a processing performance of 0.09 GB/s (20.9 GB in 218 seconds) in D3-4 and 0.42 GB/s (13.74 GB in 32.8 seconds) in D1-3. The new implementation increases the performance significantly, achieving processing speeds above 1.4 GB/s. The new data loader delivers data roughly at the rate that the GPUs can consume the data in training and is no longer a bottleneck.
The data loader is critical for this application, because the data is too large to fit in main memory (6TB) and must therefore be streamed from disk. For this reason, a smaller 330 GB dataset has been used in order to make the benchmark less time consuming, but still realistic. The goal of the AP1 loader is to be both efficient and to allow for some flexibility in changing the input data on the fly, for example via options for the patching of data in space, extra features to generate the data, and data normalization. The main challenge was to allow for arbitrary patching of the data, since extracting a single patch requires non-sequential interleaved access to the data.
The following work has been performed since the last deliverables to improve the data loader:

- Implemented a 5 stage processing pipeline (reading, feature extraction, predictor normalization, patching, and target normalization), where each stage is run in paralllel.
- After reading a file, splitting the data and processing chunks in parallel before merging them at the end.
- Forcing the processing steps to run on the CPUs, instead of on GPUs.
- Ensuring that the data reading stage is only able to use one thread. This prevents too many files to be loaded at once, which causes memory exhaustion, even with 512GB of RAM.
- Implemented a distributed pipeline using horovod, allowing 4 files to be processed and trained in parallel using MPI.

As the data loader streams data, it can be difficult to separate the time spent loading data from the time it takes to train. We therefore ran a separate data loading experiments to measure the speed of the I/O part.



| Training dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 329.83 GB | 13.74 GB | 16992 | [512,512,17] | 72 for A100; 36 for V100 |

| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| 8650611 | 0 | Quantile score (10,50,90%) | 24 days of training data; Grid split into 12 patches. |

| Inference dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 13.74 GB | N/A | 708 | [256,256,17] | 72 for A100; 36 for V100 |

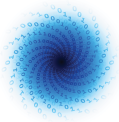
| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| 8650611 | 0 | N/A | 1 day of data; Grid split into 12 patches. |

| Data Loading dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 329.83 GB | N/A | 16992 | [256,256,17] | 72 for A100; 36 for V100 |

| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| N/A | 0 | N/A | 24 days of training data; Grid split into 12 patches. |

| Data formats | Frameworks (to be) used |
|---|---|
| NetCDF | TensorFlow2 |

The raw data discussed in this section can be found as tables in appendix 6.1.



### 4.1.2 JUWELS Booster

In the AP1, a total of 12 experiments were conducted on Juwels Booster. These 12 experiments were divided into four triplets, each using different configurations:

- Triplet 1 (experiments 1, 2, and 3): 1 Node, 1 GPU, 1 MPI Task, and the SCRATCH filesystem.
- Triplet 2 (experiments 4, 5, and 6): 1 Node, 2 GPUs, 2 MPI Tasks, and the SCRATCH filesystem.
- Triplet 3 (experiments 7, 8, and 9): 1 Node, 4 GPUs, 4 MPI Tasks, and the SCRATCH filesystem.
- Triplet 4 (experiments 10, 11, and 12): 1 Node, 4 GPUs, 4 MPI Tasks, and the CSCRATCH filesystem.

The inference phase was also tested, with a total of 3 experiments, where the total runtime and energy consumption were analyzed. Additionally, an analysis was performed on the runtime, epoch training time, and energy consumption.

#### 4.1.2.1 Training

Considering the aforementioned changes, we are interested in examining how the runtime has evolved compared to previous runs. To accomplish this, we will be analyzing the total runtime, which is mostly spitted in loading data time and training runtime. By comparing these results to those of previous runs, we can gain insight into how the changes made have impacted the performance of the application.
As seen in Figure 1, the experiments of the first trio use a single GPU, and the performance of these three is quite consistent, hovering around 870 seconds, as can be seen from the data. These are the longest runs.
By increasing the number of GPUs, a clear improvement in runtime is visible. Applications 4, 5, and 6, which use 2 GPUs, show a reduction of approximately 200 s compared to the previous case. The best performance is achieved when using 4 GPUs in parallel, resulting in a average runtime of 586.7 seconds.
When using the other file system and maintaining 4 GPUs, the runtime increases, and on average, these applications take about 717.3 s.
It is important to note that compared to the previous case, where the data loading phase was the predominant component of the total runtime, it is now clearly reduced.

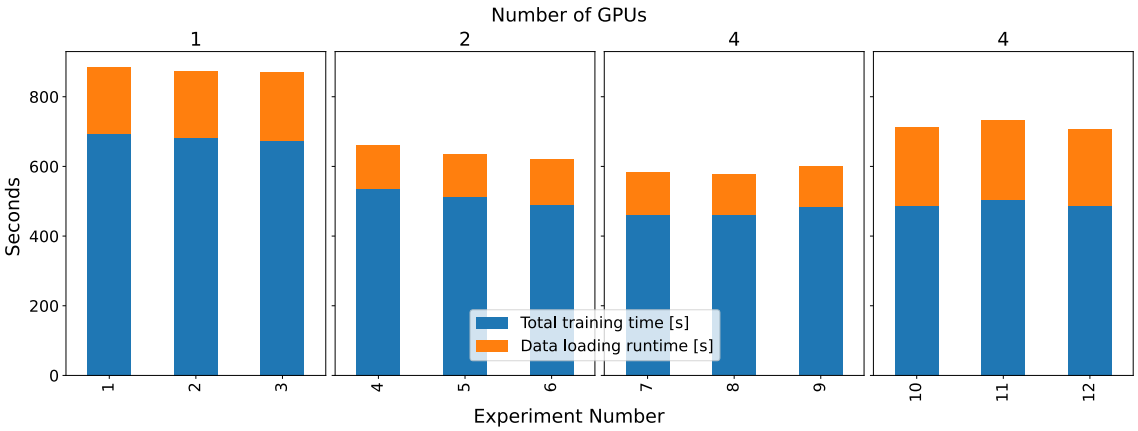



Figure 1: AP1 JUWELS Booster Runtime: Runtime and relative share for multiple experiments during the training phase. *ap1-jwb-runtime-share*

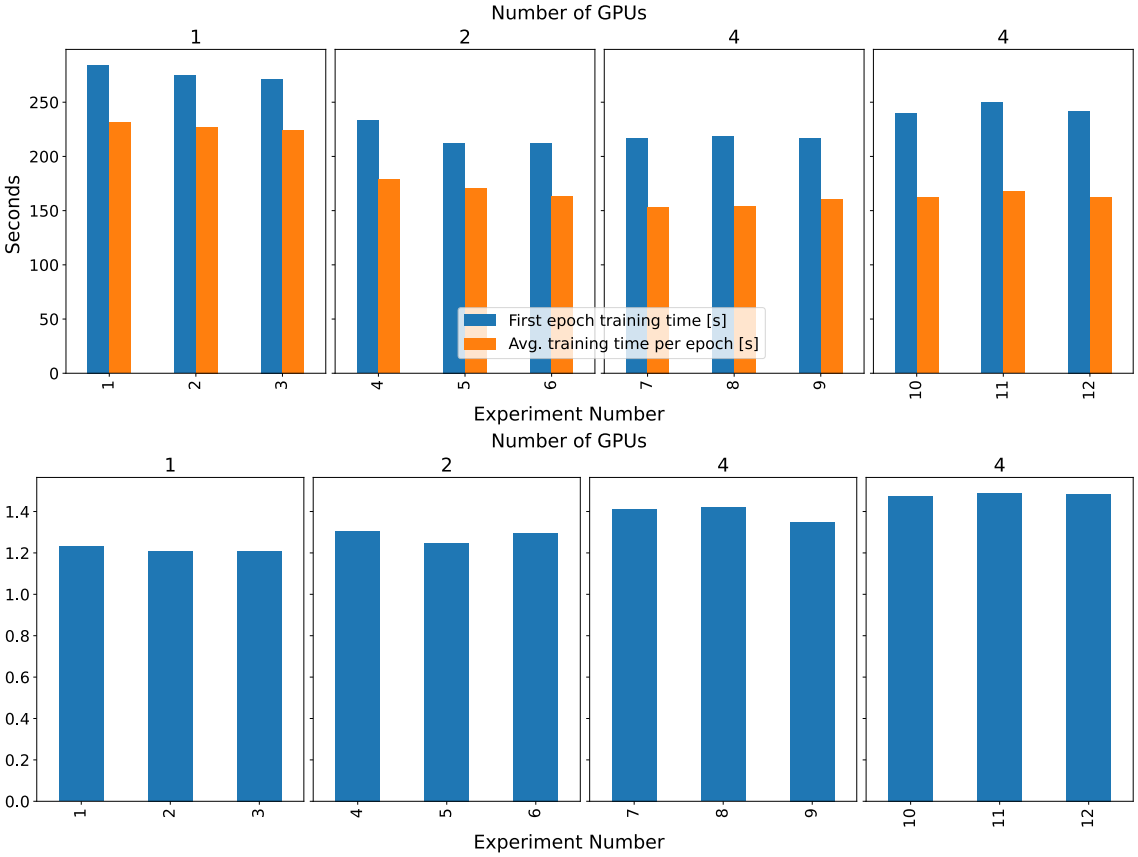



Figure 2: AP1 JUWELS Booster Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom). *ap1-jwb-epoch-time*

As the number of GPUs increases (see Figure 2), the training time for one epoch also scales accordingly. In the first three experiments, the average training time per epoch is approximately $227.5 \pm 3.5$ s, while the time to train the first epoch takes on average $1.22 \pm 0.01$ s times longer.
The average training time per epoch for the second set of three experiments is $170.8 \pm 7.4$ s, with the time to train the first epoch taking on average $1.28 \pm 0.03$ s times longer. In contrast, the third set of three experiments, which utilize 4 GPUs and achieve the best timing, has an average training time per epoch of $155.9 \pm 4.1$ s, with the time to train the first epoch taking on average $1.39 \pm 0.04$ s times longer. The last set of three experiments, which also utilize 4 GPUs but employ HPST, has an average training time per epoch of $164.2 \pm 3.0$ s, with the time to train the first epoch taking on average $1.48 \pm 0.01$ s times longer. However, the first epoch training time for this set is slightly worse, with a mean of $243.7 \pm 5.5$ s compared to the previous set.

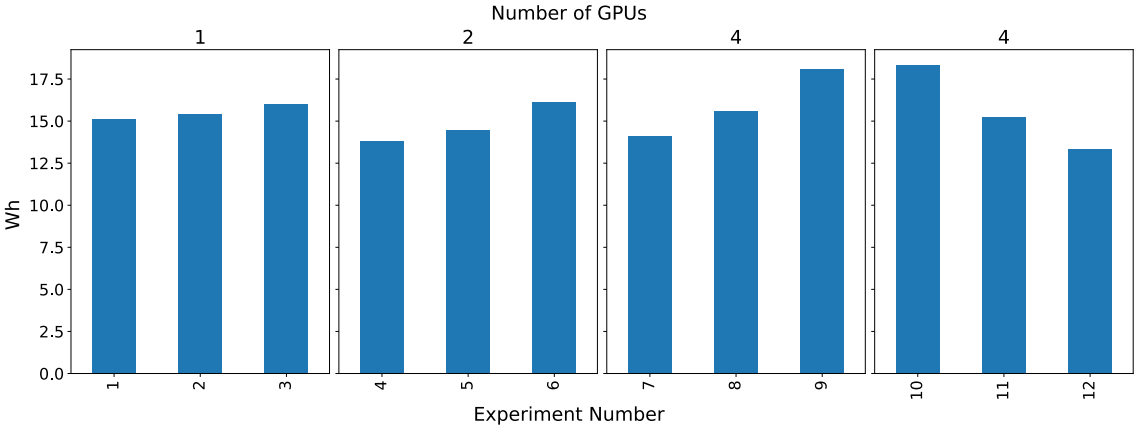



Figure 3: AP1 JUWELS Booster Energy: Total GPU energy consumption during the training phase. *ap1-jwb-energy*

The GPU energy consumption values reported in Figure 3 are specific to a single GPU and fall within a range of 14 to 18 Wh. This indicates that, overall, there was a relatively uniform energy consumption across different experimental setups.

#### 4.1.2.2 Inference

Based on data in Figure 4, we can see that the total inference time is significantly higher than the data loading overhead time. This suggests that the majority of the runtime is spent on actual inference rather than data loading.
Furthermore, we can see that there is some variation in the total inference time across the different experiments, with values ranging from 29.17 s to 34.8 s.

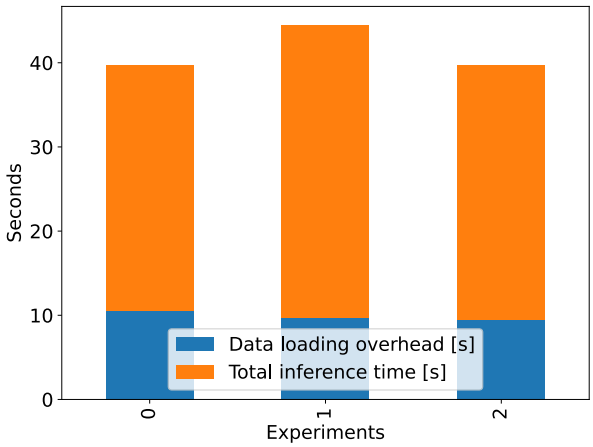



Figure 4: AP1 JUWELS Booster Inference Runtime: Runtime and relative share for multiple experiments during the inference phase. *ap1-jwb-runtime-inf*

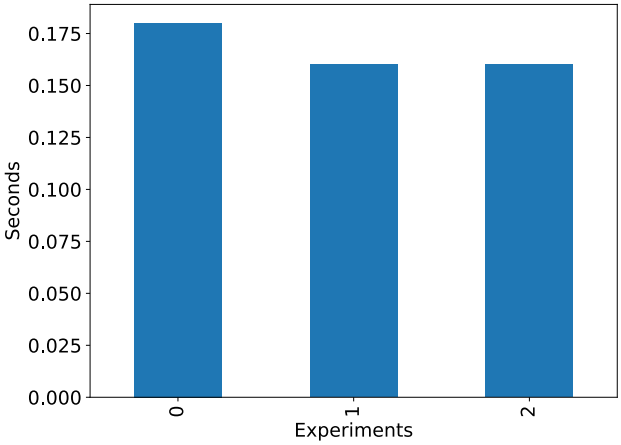
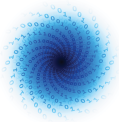



Figure 5: AP1 JUWELS Booster Energy: Total GPU energy consumption during the inference phase. *ap1-jwb-energy-inf*

Regarding the GPU energy consumption, the provided values reported in Figure 5 for the inference experiments indicate relatively low energy usage, ranging from 0.156 Wh to 0.178 Wh for a single GPU.



### 4.1.3 JUWELS Cluster

In the AP1, three experiments were conducted on Juwels Cluster, all with the same configuration of 1 node, 1 GPU, and 1 MPI Task. The following analysis focuses on runtime, epoch training time, and consumption, with only the training phase being reported.

#### 4.1.3.1 Training

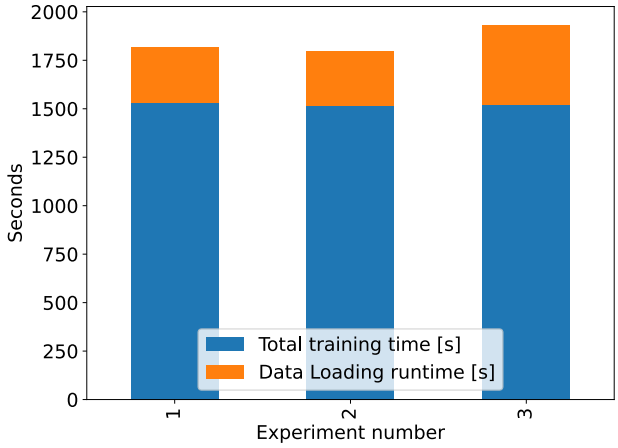



Figure 6: AP1 JUWELS Cluster Runtime: Runtime and relative share for multiple experiments *ap1-jwc-runtime-share*

As show in Figure 6, the total training time ranges from 1518.21 seconds to 1531.19 seconds, while the data loading time ranges from 279.41 seconds to 410.22 seconds. It's worth noting that the data loading time is a significant portion of the total runtime, comprising around 18-26% of the total runtime in these experiments.
The relatively small variation in total training time suggests that the training process is relatively stable and consistent across these experiments. However, the differences in data loading time may indicate that there are variations in the data processing or data access mechanisms used in these experiments.
By looking at Figure 7, we can see that the first epoch training time and the average training time per epoch are relatively close, with the average time being slightly lower. This suggests that the model training process is relatively stable, with consistent performance across epochs.
Looking at the ratio between the first epoch and average epoch training times, we see that the values are very close to 1, indicating that there is not a significant

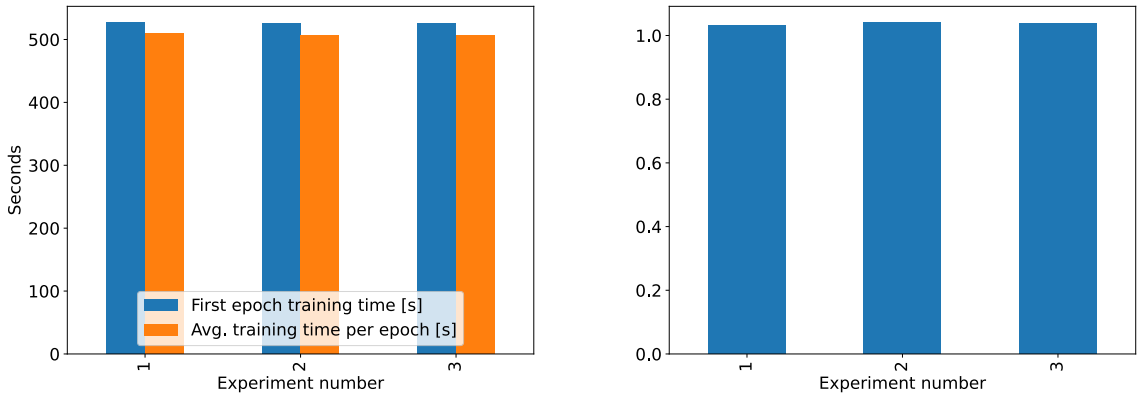



Figure 7: AP1 JUWELS Cluster Epoch Time: Comparison of time for first epoch and average time for an epoch (left); ratio of both quantities (right). *ap1-jwc-epoch-time*

difference between the two. This further supports the notion that the model training process is stable and consistent.

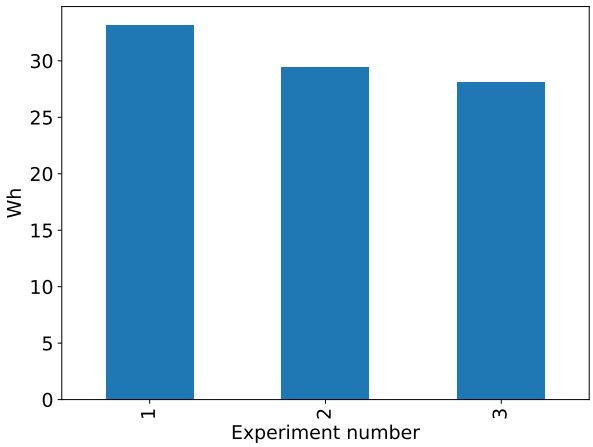
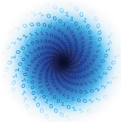



Figure 8: AP1 JUWELS Cluster Energy: Total GPU energy consumption during the training phase. *ap1-jwc-energy*

The GPU energy consumption related to the training phase (see Figure 8) varies across the different runs, but the variation between experiments is small. The values reported are 33.15 Wh, 29.48 Wh, and 28.15 Wh for the three runs, respectively, and these values appear to be relatively homogeneous.



### 4.1.4 E4 Intel System

On the E4 Intel system, six experiments were performed, all with the same configuration except for the filesystem, which was either NFS or NVMe. The following analysis focuses on runtime and energy consumption for these two different filesystems. Additionally, three experiments were performed to gather information on the inference part of the pipeline.

#### 4.1.4.1 Training

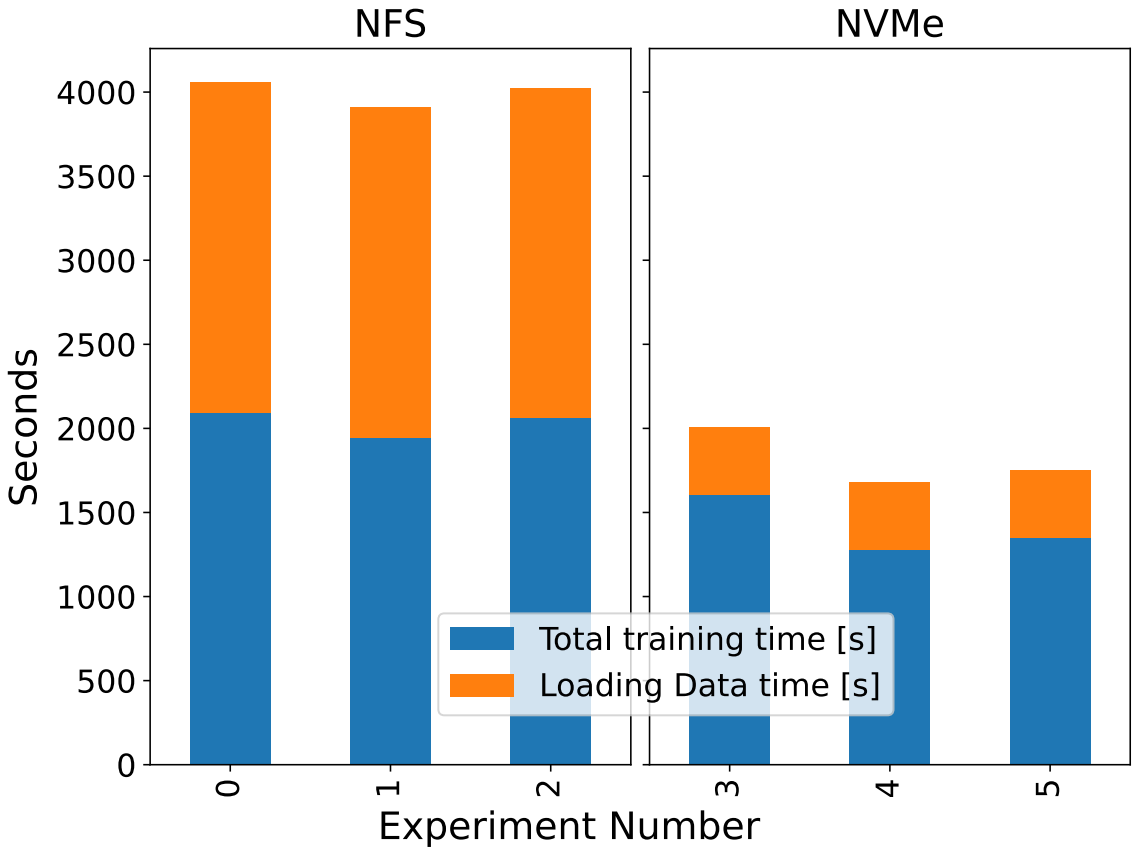



Figure 9: AP1 E4 Intel System Runtime: Runtime and relative share for multiple experiments during the training phase. *ap1-e4intel-runtime-share*

we can see that there are three experiments conducted on two different filesystems, NFS and NVMe. For the NFS filesystem, the total runtime values are reported as 2095.96 s, 1946.28 s, and 2070.76 s for experiments 0, 1, and 2, respectively. The corresponding loading data times are relatively similar, with values ranging from 1956.94 s to 1967.15 s. On the other hand, for the NVMe filesystem, the total

runtime values are reported as 1610.62 s, 1280.85 s, and 1355.70 s for experiments 0, 1, and 2, respectively, with loading data times around 400 s. While for the NFS case we see that half of the total runtime is taken by the data loading, which was a bottleneck in the previous deliverable, we obtain a very good improvement by using a more performant filesystem such as NVMe. In fact, the data loading is strongly reduced, passing from 1960 s to 400 s, which represents a reduction of almost 80% in data loading time.
Since only one epoch was used for training on the Intel system, it is not possible to compare the first and average epoch training times.

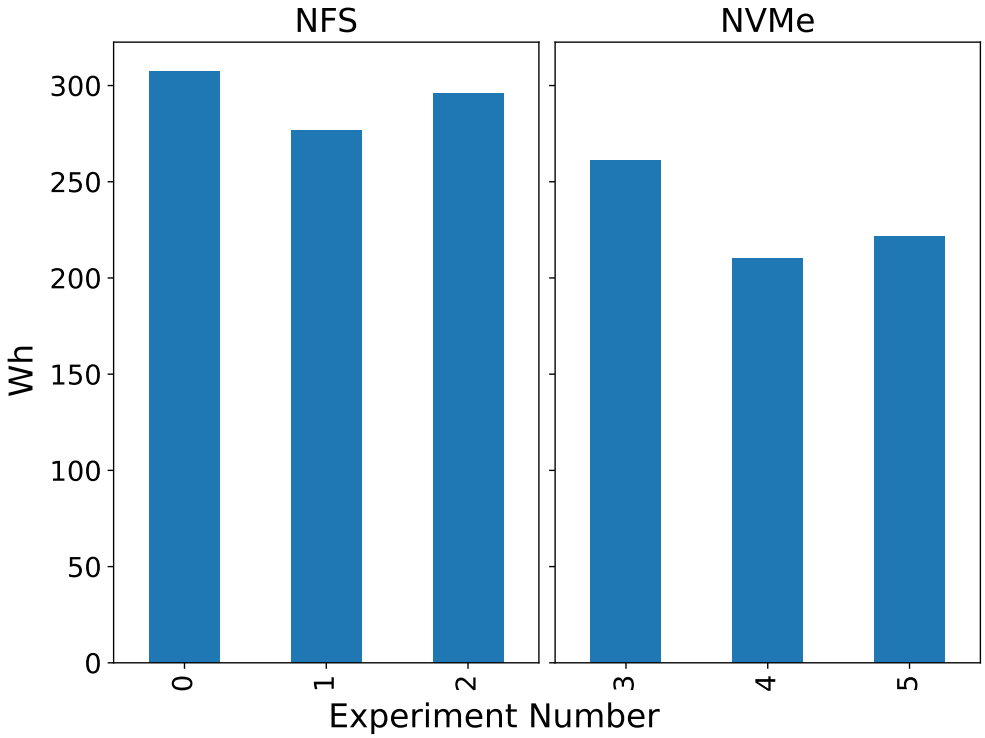



Figure 10: AP1 E4 Intel System Energy: Total energy consumption (up); peak and average node power draw (right) *ap1-e4intel-energy*

Using the NVMe filesystem not only improves the loading data time, but it also reduces the total runtime and energy consumption. In fact, as shown in Figure 10 the mean energy consumption for the NVMe experiments is $231.17 \pm 26.69$ Wh, which results lower than the one for the NFS experiments, that is $293.33 \pm 15.37$ Wh.
In Figure 11, we can observe the action values for the Intel system. The results indicate that the applications running on the NVMe filesystem have better overall performance compared to those running on NFS, as they have lower action values. On average, the action value for NFS is $2154.6 \pm 194.5$ MJs, while for NVMe it is

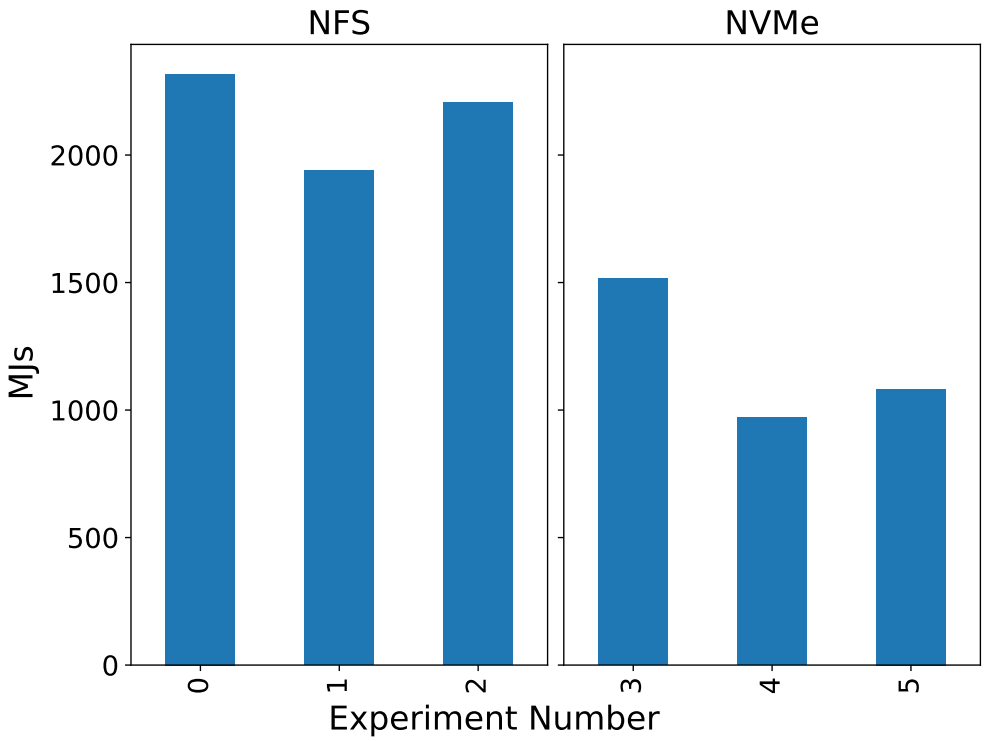



Figure 11: AP1 E4 Intel System Action: Comparison of action values for applications using different file systems during training: NVMe and NFS. *ap1-e4intel-action*

$1189.2 \pm 287.6$ MJs.
The experiment with the best performance on NVMe is experiment 1, which has the lowest action value of 970.18 MJs. On the other hand, experiment 0 on NFS has the highest action value of 2317.90 MJs, which is more than double the action value of experiment 1 on NVMe.

#### 4.1.4.2 Inference

The total inference time is relatively consistent across the different runs for both the NFS and NVMe filesystems, with values ranging from 34.24 s to 34.89 s.
However, we can see that the data loading overhead is significantly higher for the NFS filesystem, with values ranging from 80.33 s to 91.8 s, compared to the NVMe filesystem, with values ranging from 22.46 s to 22.9 s. This suggests that the data loading process may be a bottleneck when using the NFS filesystem, which could impact the overall performance of the inference process.
The energy consumption during inference varies across the different runs and filesystems. The values reported are 12.28 Wh, 11.32 Wh, and 12.89 Wh for the

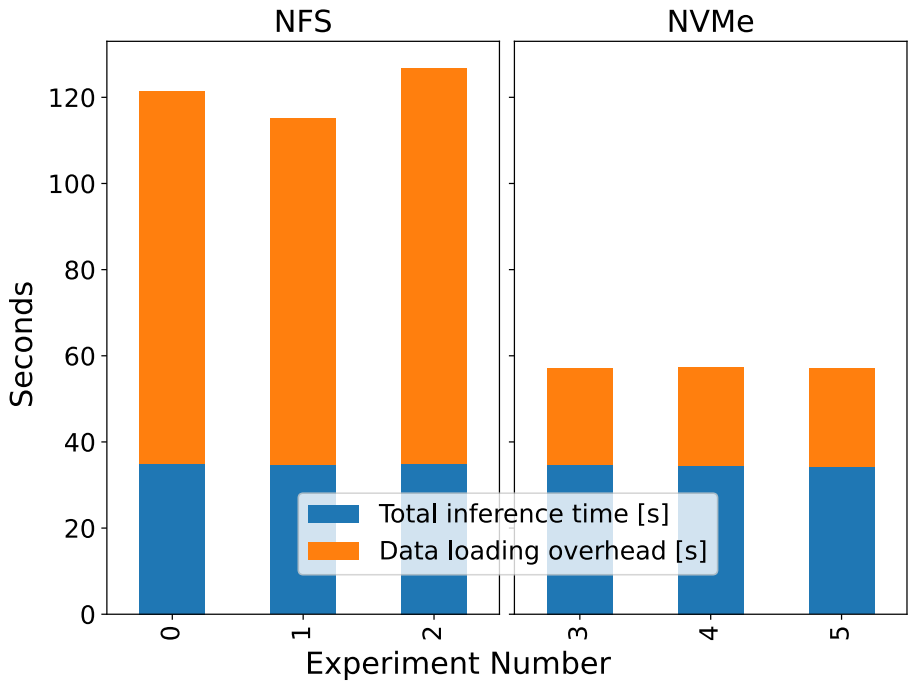



Figure 12: AP1 E4 Intel System Inference Runtime: Runtime and relative share for multiple experiments *ap1-e4i-runtime-inf*

NFS filesystem, and 3.33 Wh, 3.37 Wh, and 3.25 Wh for the NVMe filesystem, respectively. It is interesting to note that the energy consumption values for the NVMe filesystem are consistently lower than those for the NFS filesystem. This suggests that using a more performant filesystem like NVMe not only reduces the data loading overhead and total inference time but also saves energy.

As reported in Figure 14, we can see that the experiments using NVMe have a much lower action value compared to those using NFS, indicating better overall performance. On average, NFS reports an action of $5.32 \pm 0.40$ MJs, while NVMe $0.68 \pm 0.01$ MJs.

Experiment 2 on NVMe has the lowest action value of 0.67 MJs, while experiment 5 on NFS has the highest action value of 5.89 MJs, which is almost 9 times higher than the action value of experiment 2 on NVMe.

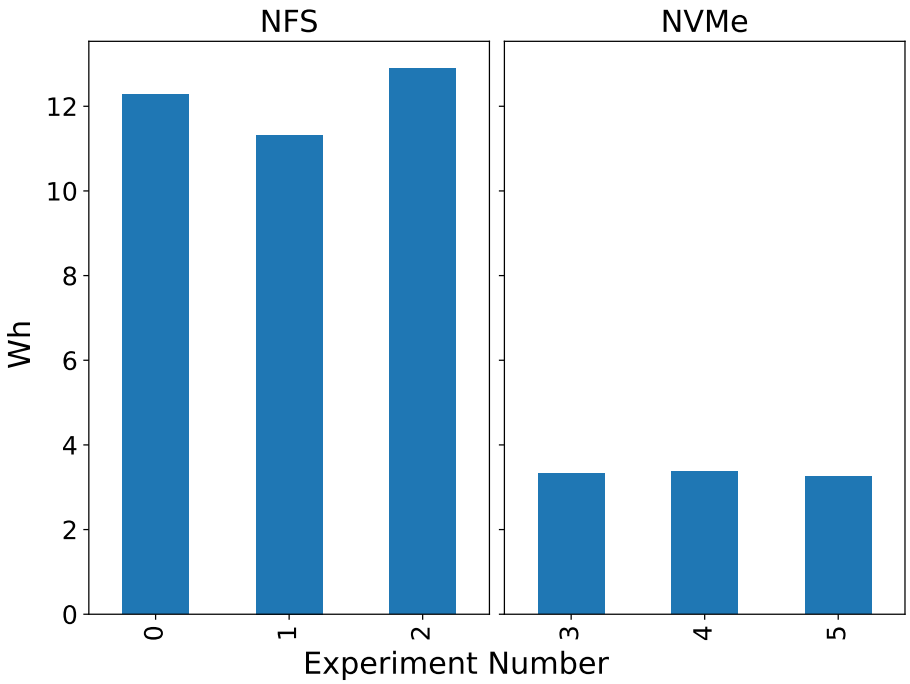



Figure 13: AP1 E4 Intel System Energy: Total node energy consumption. *ap1-e4i-energy-inf*

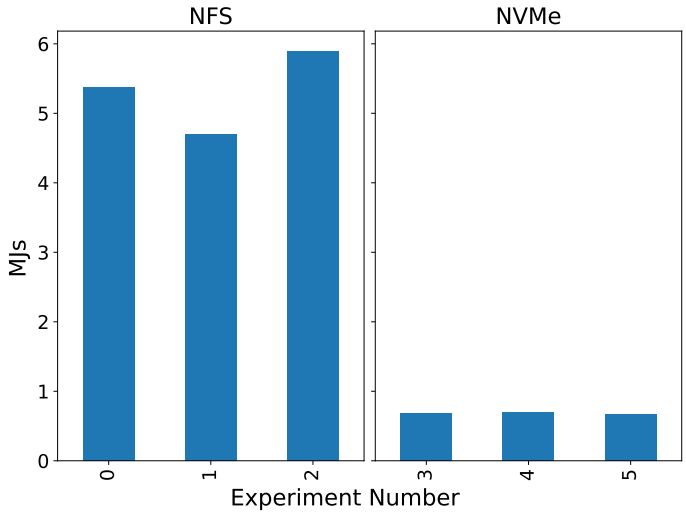
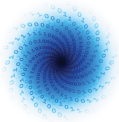



Figure 14: AP1 E4 Intel System Action: Comparison of action values for applications using different file systems during inference: NVMe and NFS. *ap1-e4i-inf-action*



### 4.1.5 E4 AMD System

On the E4 AMD system, six experiments were performed with the same configuration as the Juwels Cluster, except for the filesystem, which was either NFS or NVMe. The following analysis focuses on runtime, epoch training time, and consumption for these two different filesystems. Additionally, three tests were performed to evaluate the inference part of the pipeline.

#### 4.1.5.1 Training

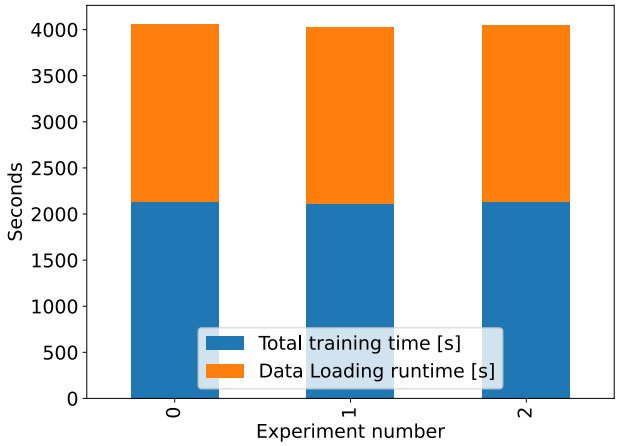



Figure 15: AP1 E4 AMD System Runtime: Runtime and relative share for multiple experiments *ap1-e4amd-runtime-share*

Based on Figure 15, we can see that the data loading time and training runtime during the training phase are relatively similar across the three experiments, with values ranging from 1914.86 s to 1920.69 s for data loading and from 2107.48 s to 2139.59 s for training runtime. It seems that the loading data time is a bottleneck in the training phase as it takes up a significant portion of the total runtime. In order to improve performance, it may be worth exploring the use of a more performant filesystem, similar to the previous case where the NVMe filesystem provided better data loading times.
Since only one epoch was used for training on the AMD system, it is not possible to compare the first and average epoch training times.
As shown in Figure 16 the energy consumption during the training phase also appears to be relatively similar, with values ranging from 359.11 Wh to 366.51 Wh across the three experiments.

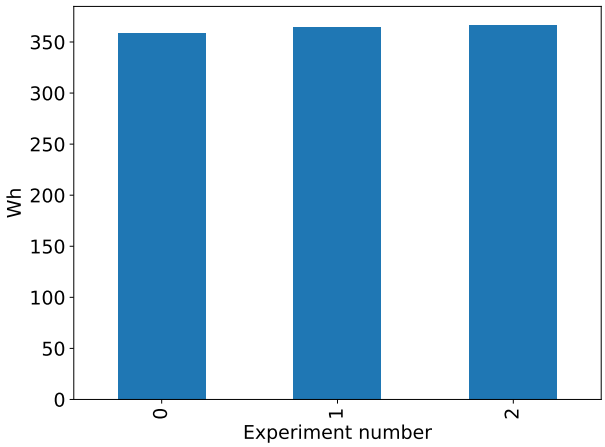



Figure 16: AP1 E4 AMD System Energy: Total node energy consumption during the training phase *ap1-e4amd-energy*

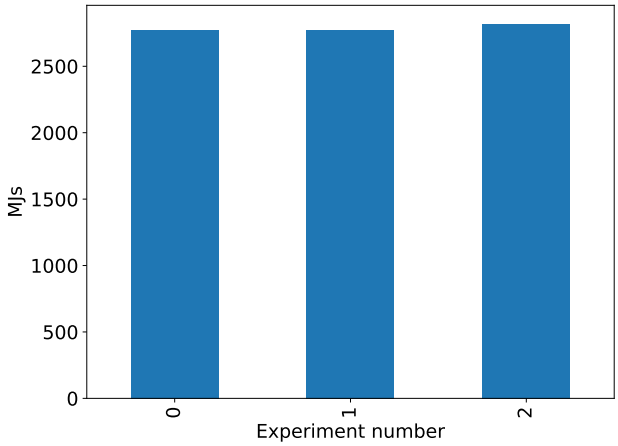



Figure 17: AP1 E4 AMD System Action: Action values (in MJs) for three applications executed on the AMD system during the training phase. *ap1-e4amd-action*

Looking at Figure 17, we see that the action values are similar, with a difference of only around 1% between the highest and lowest values. Experiment 0 reports the lowest action value of 2772.18 MJs, while experiment 2 has the highest action value of 2818.06 MJs. The average action value for NFS is 2788.27 MJs with a standard deviation of 22.11 MJs.
Compared to the results of the NVMe filesystem, the action values for NFS are higher, indicating lower overall performance.

#### 4.1.5.2 Inference

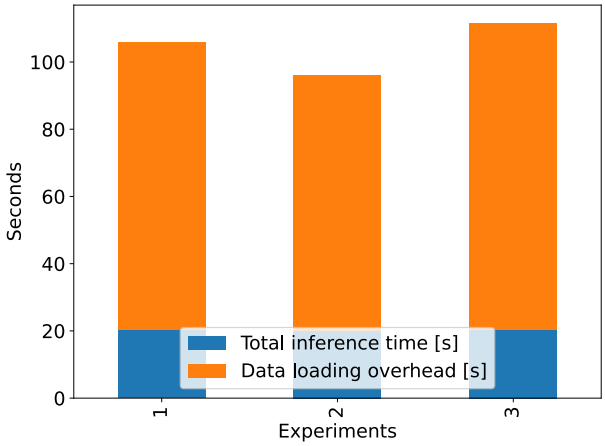



Figure 18: AP1 E4 AMD System Inference Runtime: Runtime and relative share for multiple experiments *ap1-e4a-runtime-inf*

The total inference time for the three runs varies between 20.01 s and 20.3 s, with a data loading overhead ranging from 76.04 s to 91.06 s. Based on the provided data, it may be worthwhile to further investigate the use of more performant filesystems to improve the data loading process during inference. This could potentially lead to reduced data loading overhead and improved total inference time.

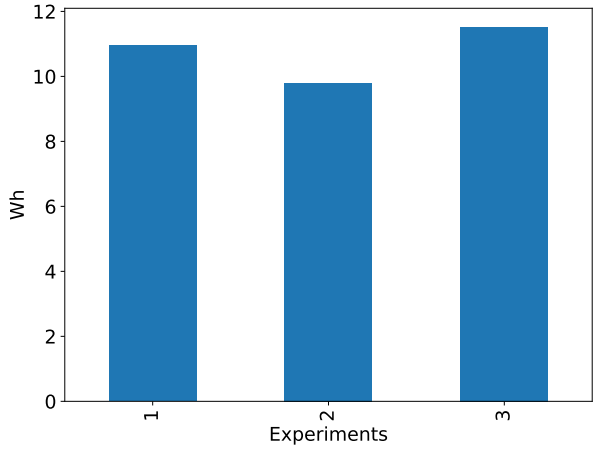



Figure 19: AP1 E4 AMD System Energy: Node energy consumption during the inference phase *ap1-e4a-energy-inf*

Regarding energy consumption, the values reported are almost constant across the three runs, with an average value of $10.75 \pm 0.88$ Wh. This suggests that the energy consumption for the inference phase is relatively stable and consistent.

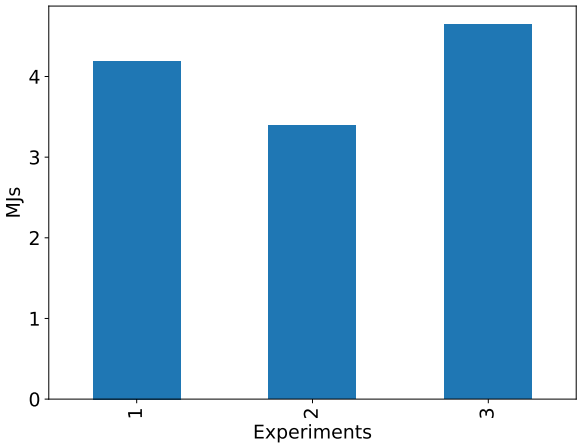
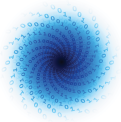
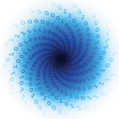



Figure 20: AP1 E4 AMD System Action: Action values (in MJs) for three applications executed on the AMD system during the inference phase *ap1-e4a-inf-action*



### 4.1.6 Results

In the context of AP1, various hardware configurations were tested, and the data reported in this second benchmarking phase shows an improvement in the time spent by the runtime in data loading compared to the previous deliverable. While the previous deliverable had a data loading time of around 60 % and 73 %, this second run showed that, with the use of better performing filesystems such as NVMe, GPFS or HPST, the time spent in data loading ranges between 18 % and 31 % of the runtime. However, when using default filesystems like NFS in E4 systems, the data loading time covers between 48 % and 50 % of the runtime. These results indicate that not only optimizations made in the data loader, but also the filesystem used plays a crucial role in the data loading process.
GPU consumption for JSC systems and node consumption for E4 systems were also recorded. It is worth noting that the consumption refers to the average consumption of the individual GPU, and the V100 GPUs in Juwels Cluster and Booster not only perform less well but also report higher consumption than the A100 GPUs.
For E4 systems, the energy consumption of the entire node was quantified, providing a more realistic view of the consumption. The results show that the Intel system outperforms the AMD system, reporting an average consumption of $293.33 \pm 15.37$ Wh and $363.51 \pm 3.90$ Wh respectively, for the same filesystem.
Taking into consideration the action metric for the E4 systems during the training phase with the same NFS filesystem, the Intel system reported an average action value of $2154.5 \pm 194.5$ MJs, while the AMD system reported an average action value of $2788.3 \pm 25.8$ MJs. Based on these values, we can conclude that the Intel system outperforms the AMD system in terms of average action.



## 4.2 AP 2

### 4.2.1 Notes

| Training dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 47.2 MB | 11.8 MB | 190k | (128100, 768) | 32 |

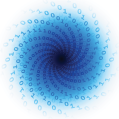
| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| 184 423 682 | 0 | cross entropy loss | Finetuning pre-trained model with small benchmark dataset |

| Inference dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 11.8 MB | - | 50k | (128100, 768) | 32 |

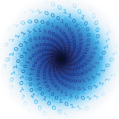
| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| 0 | 184 423 682 | - | Finetuning pre-trained model with small benchmark dataset |

| Data formats | Frameworks (to be) used |
|---|---|
| NetCDF | Pytorch 1.13 |

The task of Application 2 is to classify Tweets as "raining" or "not raining". Our current solution is based on a deep transformer based neural network that is trained on a large corpus. We focus on finetuning the model to adopt it to our specific domain (see Deliverable D1.3 for more details). A single epoch suffices to finetune our model. The model can be trained on multiple GPUs in parallel. Here, we vary the number of used GPUs to analyze the efficiency of parallelization and its efficiency on different systems. To allow for more iterations, we only use a quarter of our full training set. Note, this application was not able to participate in the first round of benchmarking (see Deliverable 3.4). Comparisons to a previous iteration of the model are therefore not possible.



First, we outline the definitions for our measured timescales as used throughout the analysis. Our "Total training time" includes model setup, actual training time and quick evaluation of model performance. Our dataset is small enough to be loaded into RAM memory. We exclude time required to load the dataset into RAM from "Total training time". We train for a single epoch. The time required for the actual training is reported as "Training time for epoch". During this training step, the data has to be provided to the model, the total time required for this process defines our "Total IO time".
Turning to evaluation, "Total IO time" measures data loading from disk only. "Total inference time" encompasses the time required to load the model and model inference. The "Total runtime" is then the sum "Total IO time" + "Total inference time".
Regarding the power consumption, different measurements were taken depending on the system. In the case of JSC, we provide power consumption of the GPU only, while in the case of the E4 machines, we report power consumption of the whole node.
Overall, training time is dominated by actual training tasks (gradient computation, backpropagation, etc.). Additional processes like model setup, IO, etc. make up less than $< 10\%$ of total training time. Therefore, most signifcant speed ups are expected from more efficient training algorithms and/or parallelization of model training.
Using multiple GPUs for training reduces training times significantly. However, scaling between execution time vs numbers of GPUs is less than ideal. For example, using 2 (4) GPUs on JUWELS Booster leads to speed ups of factors  1.6x ( 2.5x). More in-depth analysis is required to find bottlenecks and resolve them.
Looking at GPU energy consumption for training on JUWELS Booster and JUWELS Cluster, we see comparable total energy consumption when using a single GPU. Note, that a minimum number of 4x GPUs are available per node. Therefore, the additional three GPUs are then in their idle state, which is included in the total consumption. Comparing the consumption between JUWELS Booster and JUWELS Cluster when using more than a single GPU, JUWELs Booster is noticibly more efficient. This may due to the different GPU types used. JUWELS Booster consists of NVIDIA A100 GPUs vs JUWELS Cluster’s NVIDIA V100 GPUs. JUWELS Booster is $10-30\%$ faster with increasing speeds for larger numbers of GPUs used. While average power for a single GPU of JUWELS Booster is higher than for JUWELS Cluster ($\sim -10\%$), using multiple GPUs leads to lower average powers for JUWELS Booster compared to JUWELS Cluster ($\sim +10\%$). These effects compound to comparable



total power consumption 21 Wh for both systems using a single GPU and around 30% less power consumed when using multiple GPUs for JUWELS Booster compared JUWELS Cluster. Interestingly, IO takes ∼ 20% more time on JUWELS Booster compared to JUWELS Cluster. However, this has little effect on the total training times as IO makes up only 3 − 10% of the total training time.
Turning to power measurements of E4 systems. We find that their Intel based system (including an NVIDIAs A100) consumes slightly less power (∼ 5%) compared to the AMD system. However, the significantly short run time on the Intel system (0.76x) compounds to a significantly less power draw compared to the AMD system (0.74x).
Both E4's Intel setup and JUWELS Booster (1x GPU) are based on NVIDIA A100 models. Performance numbers are comparable. Therefore, difference in system setup and hardware configuration besides GPU's appear to have a negligible effect on training speeds.
The E4 Intel system equiped with an NVIDIA A100 model is significantly faster than the E4 AMD setup using an AMD Instinct MI100. The former takes on average 0.76 of the training time, 0.76 of the IO time and consumes only 0.72 of the energy of the former. Reasons for this should be investigated in the future.
Memory consumption for CPUs and GPU(s) are comparable between all tested systems (JUWELS Booster/Cluster, E4 Intel/AMD) at ∼ 4 GB and ∼ 11 GB, respectively. This makes training feasible on consumer-grade systems with a highend graphics card model. For evaluation, even just CPUs are sufficient for our dataset.

### 4.2.2 JUWELS Booster

On Juwels Booster, 10 experiments were performed for training. The number of nodes was fixed at 1, while the number of GPUs varied across the experiments. The first three runs used 1 GPU, the next three used 2 GPUs, and the last four used 4 GPUs.
For inference, three experiments were performed with fixed configurations of 1 node, 1 GPU, and 1 MPI task.

#### 4.2.2.1 Training

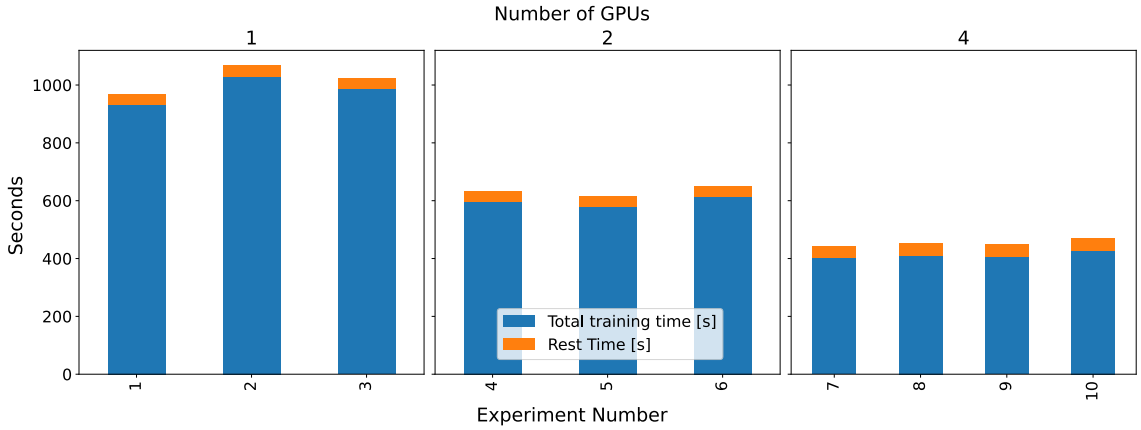



Figure 21: AP2 JUWELS Booster Runtime: Runtime and relative share for multiple experiments in the training phase. *ap2-jwb-runtime-share*

Looking at Figure 21, we can see that the total training time decreases as the number of GPUs increases. For example, with 1 GPU the training times were between 930 and 1027 seconds, while with 4 GPUs the training times ranged from 402 to 429 seconds.
On the other hand, the rest time seems to increase slightly as the number of GPUs increases. The rest time ranges from 36 to 40 seconds when using 1 or 2 GPUs, while with 4 GPUs the rest time ranges from 40 to 42 seconds.
By looking at Figure 22, as we increase the number of GPUs used in the training process, the mean energy consumption per GPU decreases. Specifically, with only one GPU, the mean energy consumption is around $22.2 \pm 2.9$ Wh, while with four GPUs, the mean energy consumption drops to around $14.2 \pm 0.4$ Wh.

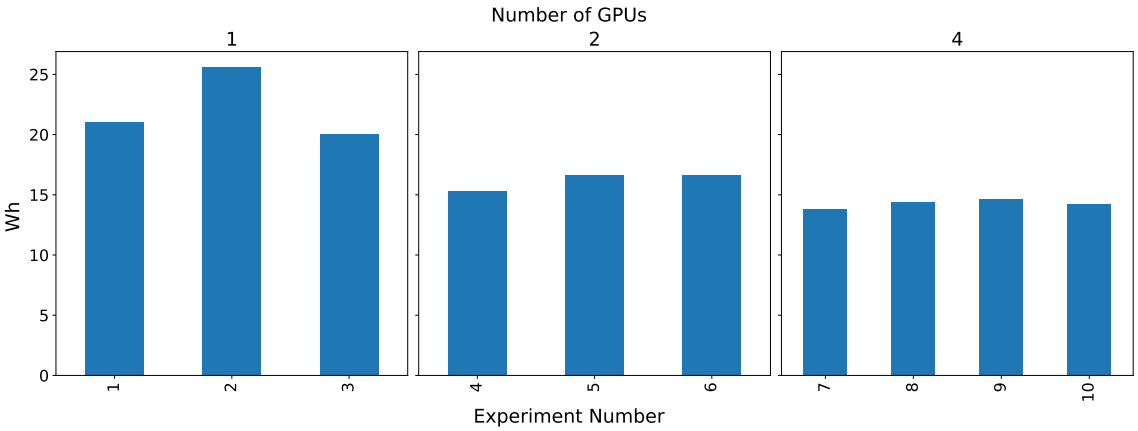



Figure 22: AP2 JUWELS Booster Energy: Total GPU energy consumption in the training phase *ap2-jwb-energy*

#### 4.2.2.2 Inference

Considering the inference phase, from Figure 23 seems that the inference time is relatively consistent across the three runs, with the total inference time ranging from approximately 18.99 seconds to 20.35 seconds. The data loading overhead also seems consistent, with values ranging from 5.75 seconds to 7.50 seconds.

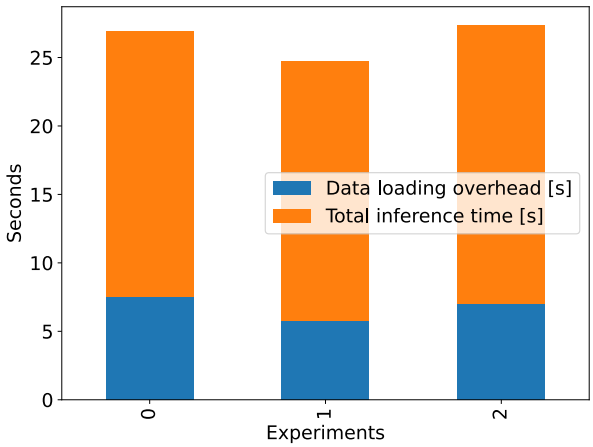



Figure 23: AP2 JUWELS Booster Inference Runtime: Runtime and relative share for multiple experiments in the inference phase. *ap2-jwb-runtime-inf*

Looking at the GPU energy consumption values during the inference phase (Figure 24), we can see that they are relatively low, with an average consumption of around $0.41 \pm 0.02$ Wh per run. This is not surprising, as the inference phase is

generally less computationally intensive compared to the training phase, and thus requires less energy.

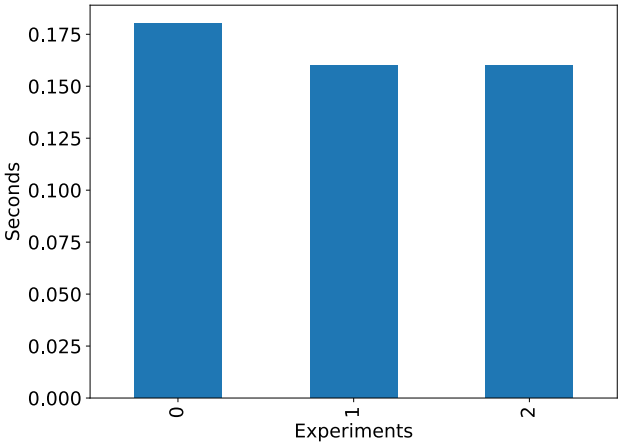
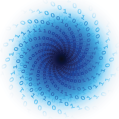



Figure 24: AP2 JUWELS Booster Energy: Total GPU energy consumption in the inference phase *ap2-jwb-energy-inf*



### 4.2.3 JUWELS Cluster

On Juwels Cluster, nine experiments were performed for training. The number of nodes and MPI tasks was fixed at 1, while the number of GPUs varied across the experiments. The first three runs used 1 GPU, the next three used 2 GPUs, and the last three used 4 GPUs.
For inference, the same number of experiments were performed with the same configurations as in the training case, including the fixed number of nodes, MPI tasks, and GPU configurations.

#### 4.2.3.1 Training

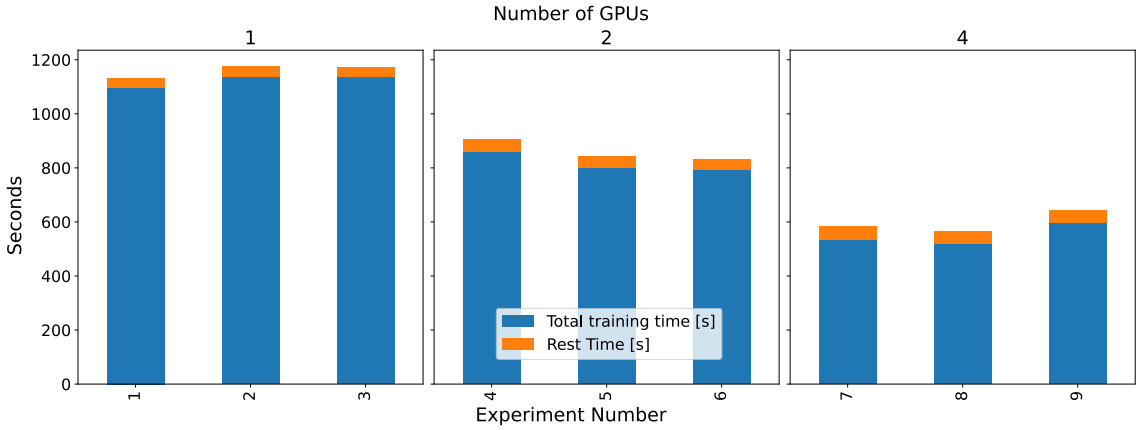



Figure 25: AP2 JUWELS Cluster Runtime: Runtime and relative share for multiple experiments in the training phase *ap2-jwc-runtime-share*

As show in Figure 25, by increasing the number of GPUs, the total training time decreases. Specifically, the total training time with 1 GPU ranges from 1.095,736 to 1.137,338 seconds, with an average of $1123.96 \pm 19.44$ s. The total training time with 2 GPUs ranges from 794.62 to 861.28 seconds, with an average of $838.97 \pm 32.86$ s. Finally, the total training time with 4 GPUs ranges from 519,437 to 598,011 seconds, with an average of $551.99 \pm 35.09$ seconds.
Additionally, we can see that the rest time (time spent outside of training) generally decreases as the number of GPUs increases, although there are some exceptions. The rest time with 1 GPU ranges from 33.51 to 39.76 seconds, with an average of $35.93 \pm 2.76$ seconds. The rest time with 2 GPUs ranges from 38.01 to 44.98 seconds, with an average of $42.01 \pm 3.66$ seconds. Finally, the rest time with 4 GPUs ranges from 44.50 to 50.78 seconds, with an average of $47.02 \pm 3.13$ seconds.
Overall, these results suggest that using more GPUs can significantly reduce the

total training time, but may not always result in a proportionate reduction in rest time.

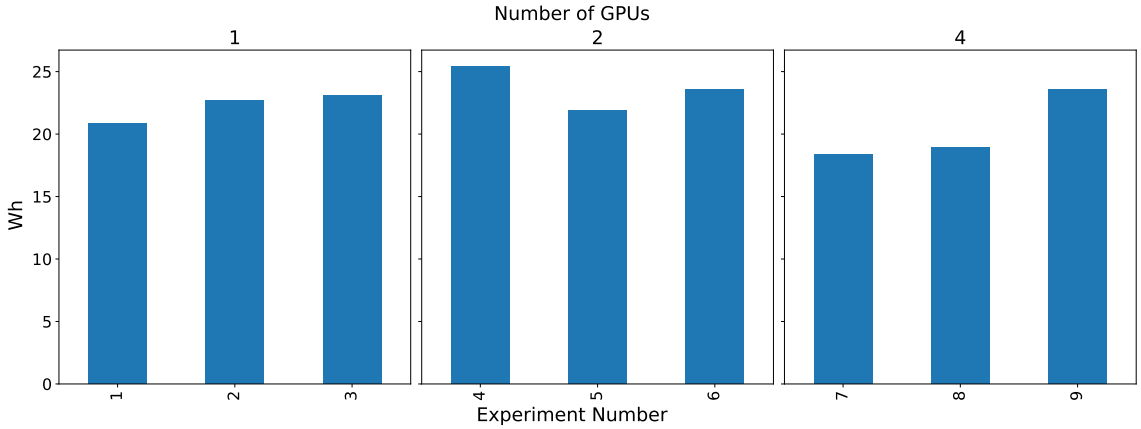



Figure 26: AP2 JUWELS Cluster Energy: Total GPU energy consumption in the training phase. *ap2-jwc-energy*

Looking at Figure 26, we can see that the value of each GPU vary between 18.36 and 25.45 Wh. Based on the GPU energy consumption data, it seems that the energy consumption is relatively consistent across different configurations. In the 1 GPU configuration, the energy consumption ranges from 20.88 to 23.07 Wh with an average of $22.22 \pm 1.00$ Wh. In the 2 GPU configuration, the energy consumption averages $23.30 \pm 1.32$ Wh. Finally, in the 4 GPU configuration, the average power consumption is 20.28 Wh.

#### 4.2.3.2 Inference

In Figure 27, the data loading overhead appears to be consistent across all experiments and does not seem to be affected by the number of GPUs used. However, the total inference runtime displays a highly variable behavior, with a general trend of increasing mean inference time as the number of GPUs increases. Specifically, the mean inference time increases from $18.63 \pm 5.17$ seconds in the 1-GPU configuration to $24.32 \pm 1.74$ seconds in the 4-GPU configuration.
Figure 28 shows that the energy consumption increases with the number of GPUs used for inference, with the highest consumption being in the 4 GPU configuration. However, it shows also that the lowest consumption per GPU being in the 2 GPU configuration.
We can also compare the energy consumption with the data loading overhead and total inference time. It appears that there is not a clear correlation between energy consumption and data loading overhead or total inference time.

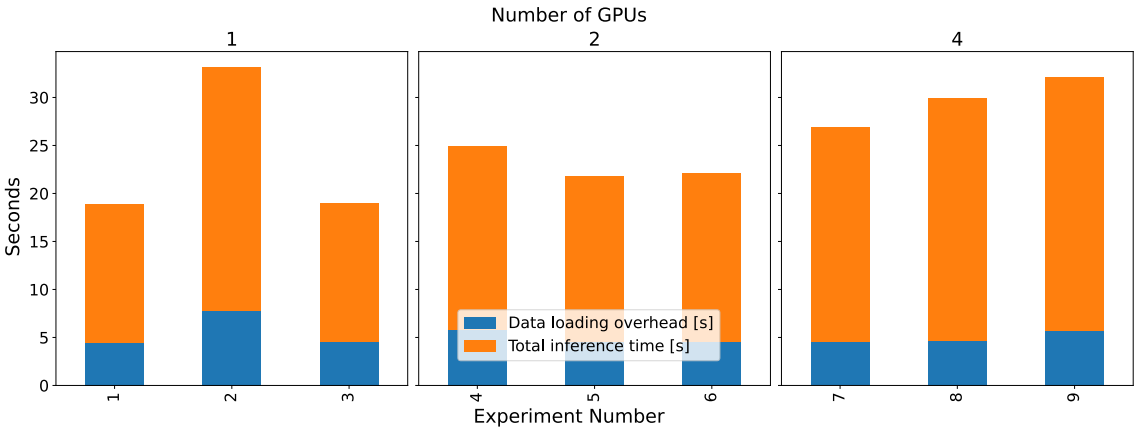



Figure 27: AP2 JUWELS Cluster Inference Runtime: Runtime and relative share for multiple experiments in the inference phase *ap2-jwc-runtime-inf*

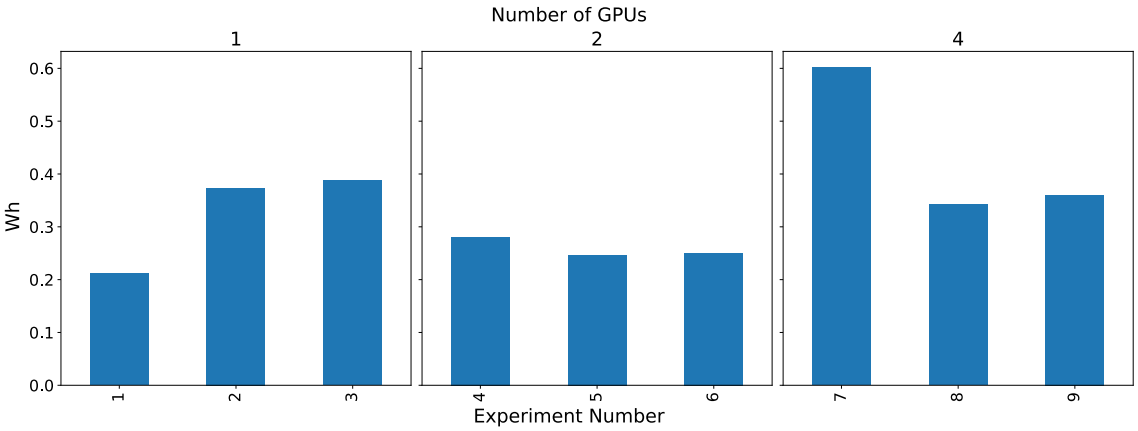
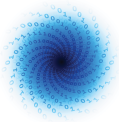



Figure 28: AP2 JUWELS Cluster Energy: Total GPU energy consumption in the inference phase. *ap2-jwc-energy-inf*

### 4.2.4 E4 Intel System

In AP2, three experiments were conducted to evaluate the performance of training on the E4 Intel System. These three runs had the same configuration, using 1 node, 1 GPU, and 1 MPI task. The same configuration and number of tests were used for inference.

In the following, we report an analysis on runtime, epoch training time, and consumptions.



#### 4.2.4.1 Training

The total runtime of the training phase is reported in Figure 29. The total training time for the three runs was 1002.64 seconds, 1139.48 seconds, and 1019.96 seconds. The average total training time was $1054.36 \pm 68.11$ s, while the average rest time amounts to $26.00 \pm 4.41$ s
These results show that the total training time varies among the three runs, with a standard deviation of around 6.5% of the average value. The rest time, on the other hand, shows less variability.

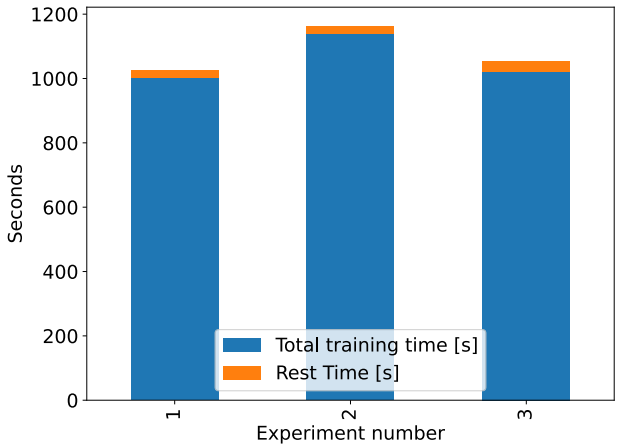



Figure 29: AP2 E4 Intel System Inference Runtime: Runtime and relative share for multiple experiments in the training phase. *ap2-e4i-runtime-run*

In Figure 30, the energy consumption values for the entire node during the training process is reported.
For the three training runs, the energy consumption values are 153.01 Wh, 169.96 Wh, and 160.50 Wh, respectively, and so the energy consumption values appear to be relatively consistent, with the standard deviation likely to be low. The average energy consumption amounts to $161.16 \pm 8.65$ Wh.
After analyzing the action values for the three experiments in Figure 31, it can be observed that there is some variability among them. Experiment 1 has the highest action value of 697.19 MJs, while Experiment 0 has the lowest action value of 552.29 MJs, and Experiment 2 has an action value of 589.33 MJs, which is in between the other two. The average action value across the three experiments is $612.94 \pm 75.28$ MJs.

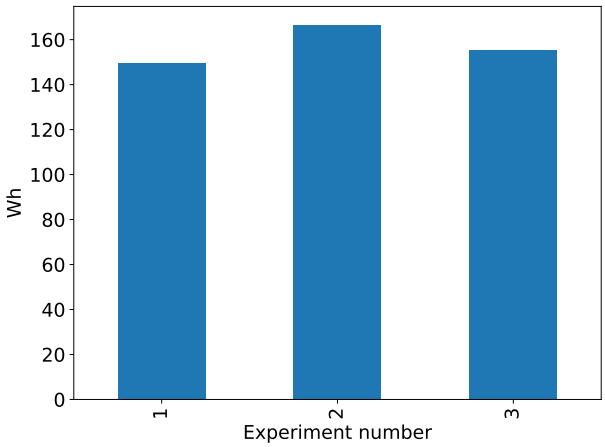



Figure 30: AP2 E4 Intel System Energy: Total node energy consumption in the training phase *ap2-e4i-energy*

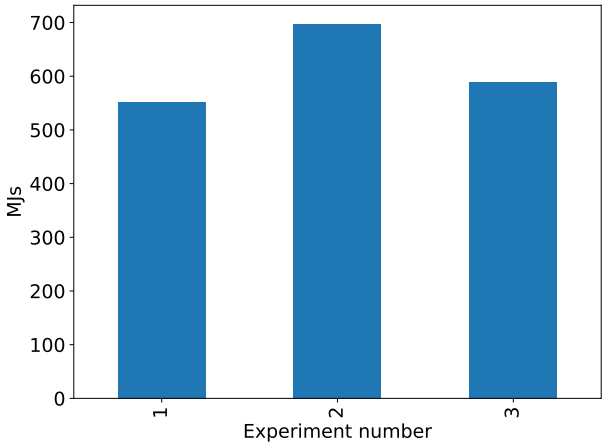



Figure 31: AP2 E4 Intel System Action: Action values (in MJs) for three applications executed on the Intel system during the training phase. *ap2-e4i-action*

#### 4.2.4.2 Inference

As show in Figure 32, the data loading overhead is relatively constant across all three experiments, with values ranging from 3.71 to 3.98 seconds. On the other hand, the total inference time varies from 11.03 to 12.70 seconds, with a mean inference time $11.60 \pm 0.80$ s. This indicates that the inference time is relatively stable across the experiments.

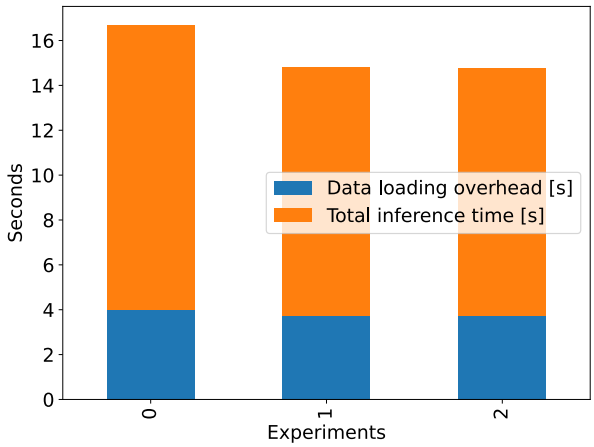



Figure 32: AP2 E4 Intel System Inference Runtime: Runtime and relative share for multiple experiments in the inference phase. *ap2-e4i-runtime-inf*

Looking at the energy consumption measurements, shown in Figure 33, we can see that they are relatively close to each other, ranging from 2.09 to 2.16 Wh, with a mean energy consumption of $2.12 \pm 0.04$ Wh. This suggests that also the energy consumption is consistent across the experiments.

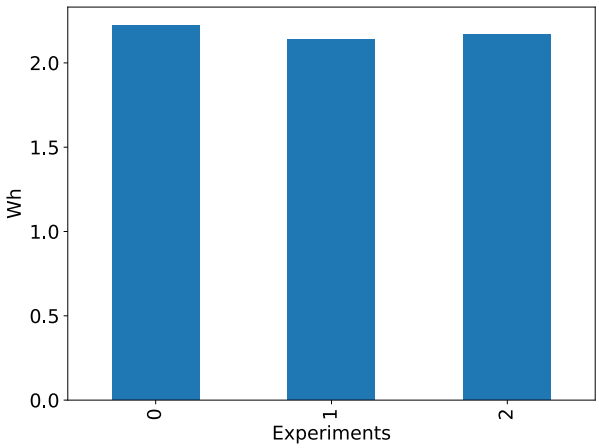



Figure 33: AP2 E4 Intel System Energy: Total node energy consumption in the inference phase. *ap2-e4i-energy-inf*

By looking at the action represented in Figure 34, it appears that all three experiments have relatively similar performance, with Experiment 0 having the highest value of 0.13 MJs and Experiment 1 having the lowest value of 0.11 MJs. The differ-

ence in action values between the experiments is relatively small, with a range of only 0.02 MJs.

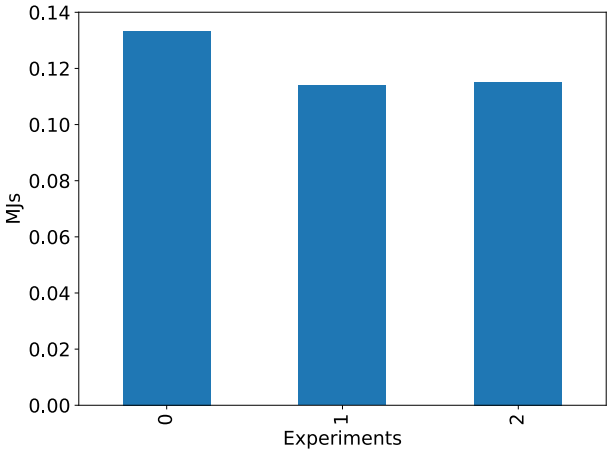
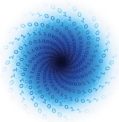



Figure 34: AP2 E4 Intel System Action: Action values (in MJs) for three applications executed on the Intel system during the inference phase. *ap2-e4i-action-inf*



### 4.2.5 E4 AMD System

In the AP2, three experiments were conducted on the E4 AMD System. These three runs had the same configuration, using 1 node, 1 GPU, and 1 MPI task. The same configuration and number of tests were used for inference.

#### 4.2.5.1 Training

The total training time for this system results to be longer than in the Intel case, with values ranging from 1364.05 to 1399.14 s, while the rest time values range from 30.63 to 30.76 seconds. As shown in Figure 35, these three experiments looks consistent and homogeneus between each other, both in the total training time and in the rest time.

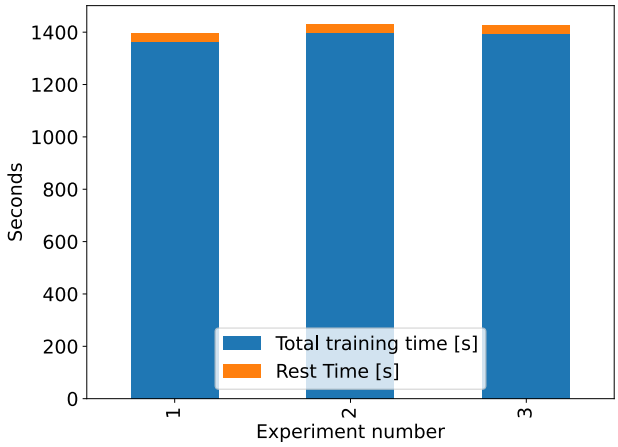



Figure 35: AP2 E4 AMD System Runtime: Runtime and relative share for multiple experiments in the training phase. *ap2-e4amd-runtime-share*

Also in the energy consumption case, see Figure 36, the experiment shows very little variation, with values ranging from 214.43 to 217.85 Wh.
The action metric was used to evaluate the performance of the AMD cluster, as shown in Figure 37. The results indicate that all three experiments have similar action values, with Experiment 0 recording the lowest value of 1076.6 MJs, and Experiment 2 recording the highest value of 1092.71 MJs. The range of action values between the experiments is relatively small, with only a 15.04 MJs difference.

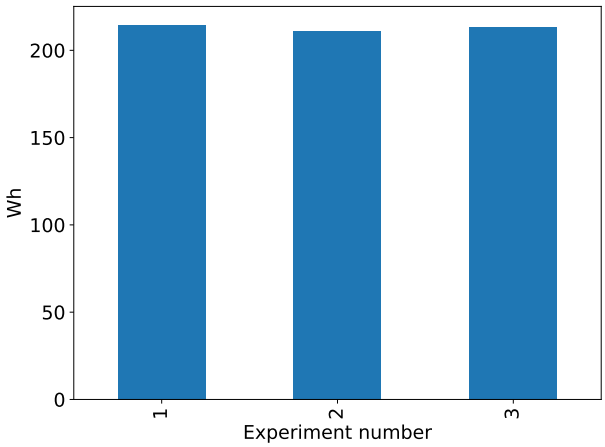



Figure 36: AP2 E4 AMD System Energy: Total node energy consumption in the training phase. *ap2-e4amd-energy*

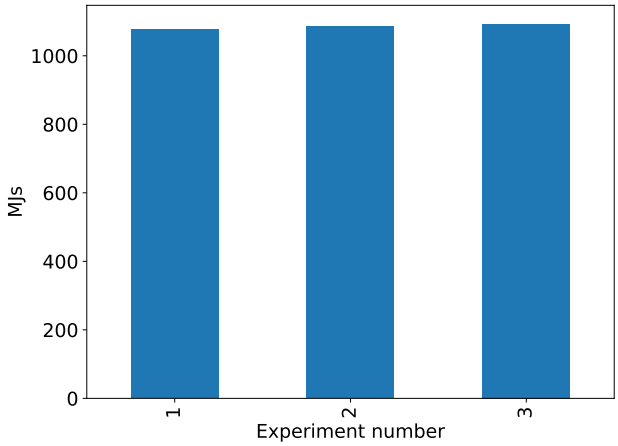



Figure 37: AP2 E4 AMD System Action: Action values (in MJs) for three applications executed on the AMD system during the training phase. *ap2-e4amd-action*

#### 4.2.5.2 Inference

For what concerns the inference (see Figure 38), by analyzing the total runtime we have seen that the average data loading overhead covers $3.94 \pm 0.52$ s of the total runtime, while the average inference time takes $18.23 \pm 5.64$ s. The data loading overhead appears consistent across all runs, while the total inference time shows significant variation, with experiments 2 and 3 having similar times ($15.30 \pm 0.01$ s)

and experiment 1 taking 61% more time (24.43 s). These findings suggest that the data loading overhead is relatively stable across experiments, while the total inference time is subject to greater variation.

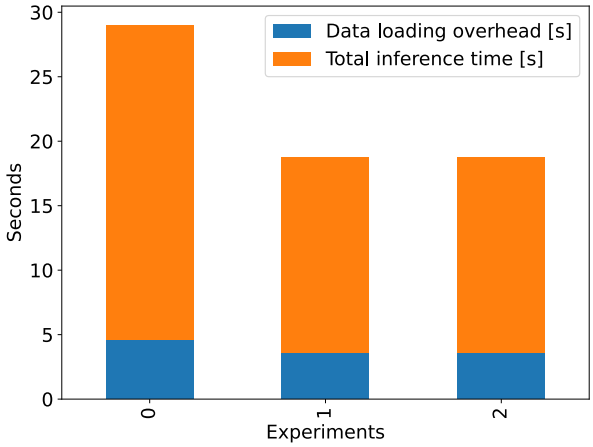



Figure 38: AP2 E4 AMD System Inference Runtime: Runtime and relative share for multiple experiments in the inference phase *ap2-e4a-runtime-inf*

As shown in Figure 39, the energy behavior is somewhat consistent with the inference time behavior. This suggests that longer inference times require more energy consumption. Specifically, the first experiment consumed 3.45 Wh of energy, while the other two experiments had lower energy consumption of 2.41 Wh and 2.45 Wh, respectively.

By looking at Figure 40, we see that experiment 0 has the highest action value of 0.38 MJs, while experiment 1 has the lowest action value of 0.16 MJs. Experiment 2 has an action value of 0.17 MJs, which is very close to the value of experiment 1. Since the action values are very small, we can conclude that the system has performed well and efficiently for all three experiments. However, experiments 1 and 2 have lower action values, indicating better overall performance and energy efficiency.

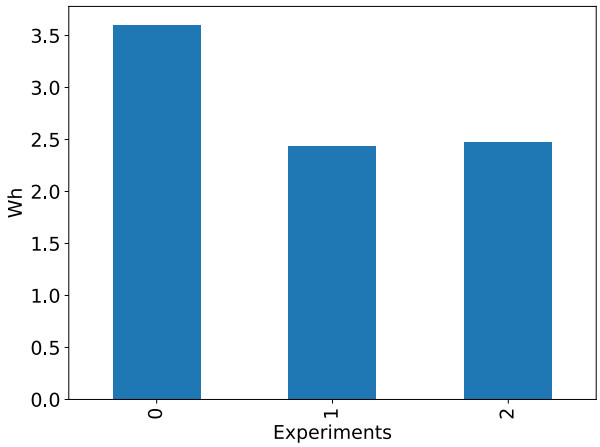



Figure 39: AP2 E4 AMD System Energy: Total node energy consumption in the inference phase *ap2-e4a-energy-inf*

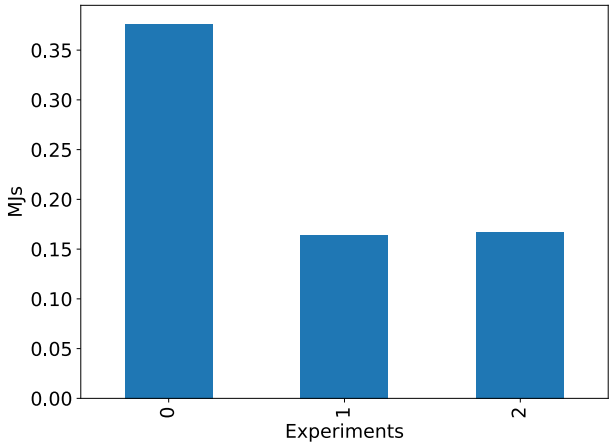
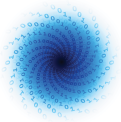


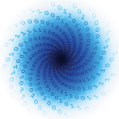

Figure 40: AP2 E4 AMD System Action: Action values (in MJs) for three applications executed on the AMD system during the inference phase. *ap2-e4a-action-inf*

### 4.2.6 Results

In the JSC systems, multiple experiments were conducted with different numbers of GPUs on both a booster and a cluster. The findings indicate that there is a notable enhancement in runtime with the use of more GPUs. Specifically, compared to the single-GPU configuration, employing four GPUs led to a reduction in runtime from around $1019.6 \pm 49.9$ to $453.9 \pm 12.1$ s (Booster case) and from $1159.2 \pm 26.1$ to



550.0 ± 42.1 s (Cluster case), resulting in an average of 2.1 times faster performance than the case with only one GPU.
In the E4 machines, single-GPU experiments were performed, and the results consistently showed that the Intel system outperformed the AMD system in both training and inference.
The measurements displayed relatively consistent values for both individual GPUs and the entire node in terms of power consumption. During the training process, the JSC system showed an average consumption per single GPU of 17.24 ± 3.83 Wh and 22.05±2.30 Wh for Juwels Booster and Cluster, respectively. On the other hand, the average power consumption of the entire node was 157.17 ± 8.59 Wh for the Intel system and 212.84 ± 1.77 Wh for the AMD system in the case of E4 systems. In terms of the action metric, which combines both runtime and consumption, the Intel system had a lower action value than the AMD system, with values of 612.9 ± 75.3 and 1085.0 ± 8.1 MJs, respectively, indicating that the Intel system was more performant.



## 4.3 AP 3

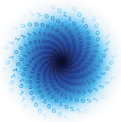
### 4.3.1 Notes

| Training dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 60 GB | 4.2 GB | 2 984 960 | (17), (137, 27), (138, 2), (138, 1) | 512 |

| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| 261 515 | 0 | MSE (multiple output vectors) | Model not trained to convergence for cost reasons (only 5 epochs), 50 epochs required |

| Data formats | Frameworks (to be) used |
|---|---|
| NetCDF | TensorFlow 2.X |

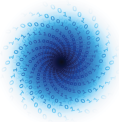
The dataset has remained the same since the previous benchmarking effort. Developments have focusssed on improving the model architecture. We have settled on a predominantly RNN architecture, specifically using LSTM blocks to propagate information in the vertical dimension. This has a very small footprint in number of trainable parameters, $O(10^5)$, but surpasses the final metric scores of the previously explored architecture, which featured convolutional and self-attention layers. Note, with the chosen batch size for this benchmarking effort only a small fraction of the GPU memory is used on most system. This batch size was chosen to optimise final metrics. Results in the below highlight that further effort should be made to explore large batch sizes without degrading final training metrics.
Deliverables D1-3 and D3-4 highlighted that loading data from disk was a bottleneck on some of the machines. In this deliverable we have added a data loading only test, where no training is done, to better understand the limitting effect of data loading. As with the previous deliverable we run experiments both using CPU memory caching of the dataset and disabling it. Whilst this dataset can be cached in memory, the full dataset is $O(200Gb)$ and therefore would not necessarily fit into CPU memory and must be streamed. A cache-free solution is therefore highly desirable.



For AP3 experiments were performed with different flags supplied to the application. The configurations are:

- *None*: Default version without special flags
- `--nocache`: Avoid using TensorFlow dataset cache
- `--dl_test`: Iterate through the training dataset without training the model, to test data-loading capabilities.
- `CSCRATCH`: Using “fast” disk on either Julich or E4 hardware, as opposed to the default location.

Raw data for graphs and discussions of this section is listed in appendix 6.3.



### 4.3.2 JUWELS Booster

In the Juwels Booster, 12 experiments have been conducted, divided into four triplets with different flags and configurations. The number of GPUs and MPI tasks used in these experiments was set to 1. The configuration for each triplet is as follows:

- Triplet 1: –nocache flag and SCRATCH filesystem.
- Triplet 2: no flag and SCRATCH filesystem.
- Triplet 3: –dl_test –nocache flag and SCRATCH filesystem.
- Triplet 4: –nocache flag and CSCRATCH fast disk.

The runtime of the inference phase has also been reported, with three experiments performed using this system.

#### 4.3.2.1 Training

Figure 41 depicts the total training time and rest time for different runs of the AP3 with different flags and disks.
For the runs with the nocache flag, the average total runtime is $1614.81 \pm 0.02$ s, while the rest time accounts for about 0.2% of the total runtime. This set of runs is the longest among the experiments using the SCRATCH disk, as this version avoids using TF dataset cache.
Moving to the Default version, we observe an average total runtime of $1155.16 \pm 34.69$, which shows a decrease of approximately 28.4% with respect to the previous case.
The runs with the –dl_test flag test the data loading capabilities and also use the –nocache flag. The average runtime is $1009.71 \pm 0.01$ s, which represents approximately 62.5% of the training runtime in the case with the –nocache flag.
The last set of runs uses the CSCRATCH fast disk with the –nocache flag. These runs take more time with respect to the SCRATCH disk, with an average runtime value of $1916.34 \pm 0.16$ s, representing an increase of approximately 18.7% with respect to the runs using the –nocache flag and the SCRATCH disk.
Figure 42 shows the comparison between the first epoch training time and the average training time per epoch for the same runs as described before. It is notable that the training times for the first epoch are very similar to the times for the later epochs in the experiments with the –nocache flag, namely triplets 1, 3, and 4. These triplets exhibit a value very close to 1 in the plot, which depicts the ratio between

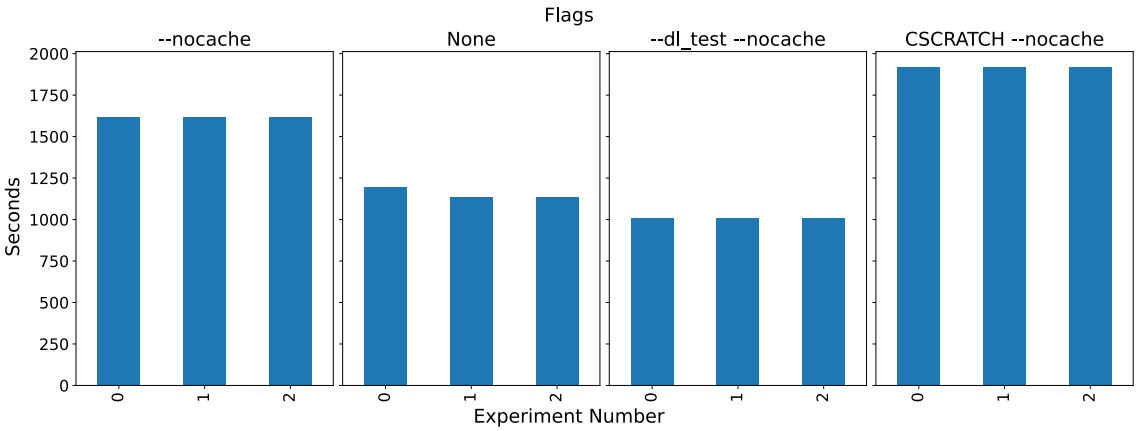



Figure 41: AP3 JUWELS Booster Runtime: Runtime for multiple experiments during the training phase. *ap3-jwb-runtime-share*

the two times. On the other hand, the runs without the flag show a considerable difference between the first epoch and the average epoch training time, which averages to $143.90 \pm 3.13$ s. In the runs without the –nocache flag, the TensorFlow dataset cache is used, which can lead to some overhead during the first epoch.
The plot representing GPU power consumption (Figure 43) follows the same trend as the training runtime plot, indicating a direct proportionality between the two quantities. For the runs using the –nocache flag, the average GPU power consumption is around $28.47 \pm 1.05$ watts, with a slight variation among the different runs. However, this value decreases significantly in the case without the flag, reaching an average consumption of $21.47 \pm 0.86$ watts, with a decrease of approximately. This can be explained by the fact that using the cache implies a higher workload for the GPU, hence a higher power consumption due to the overhead. Regarding the data loading test, the average consumption is around $16.54 \pm 0.27$ watts, with a small variation among the different runs. Finally, for the three runs using the CSCRATCH disk, the average GPU power consumption is around $36.58 \pm 0.39$ watts, which is higher than the consumption observed in the previous cases.

#### 4.3.2.2 Inference

For the inference test, three runs were performed without any flags.
A show in Figure 44, the data loading overhead time for the three runs ranges from 1.04 to 1.13 seconds, with an average value of around 1.1 seconds. The total inference time for the three runs is very consistent, ranging from 42.25 to 42.27 seconds, with a very small variation of only 0.02 seconds. The rest time for the runs is also very similar, ranging from 1.21 to 1.22 seconds. This rest time represents

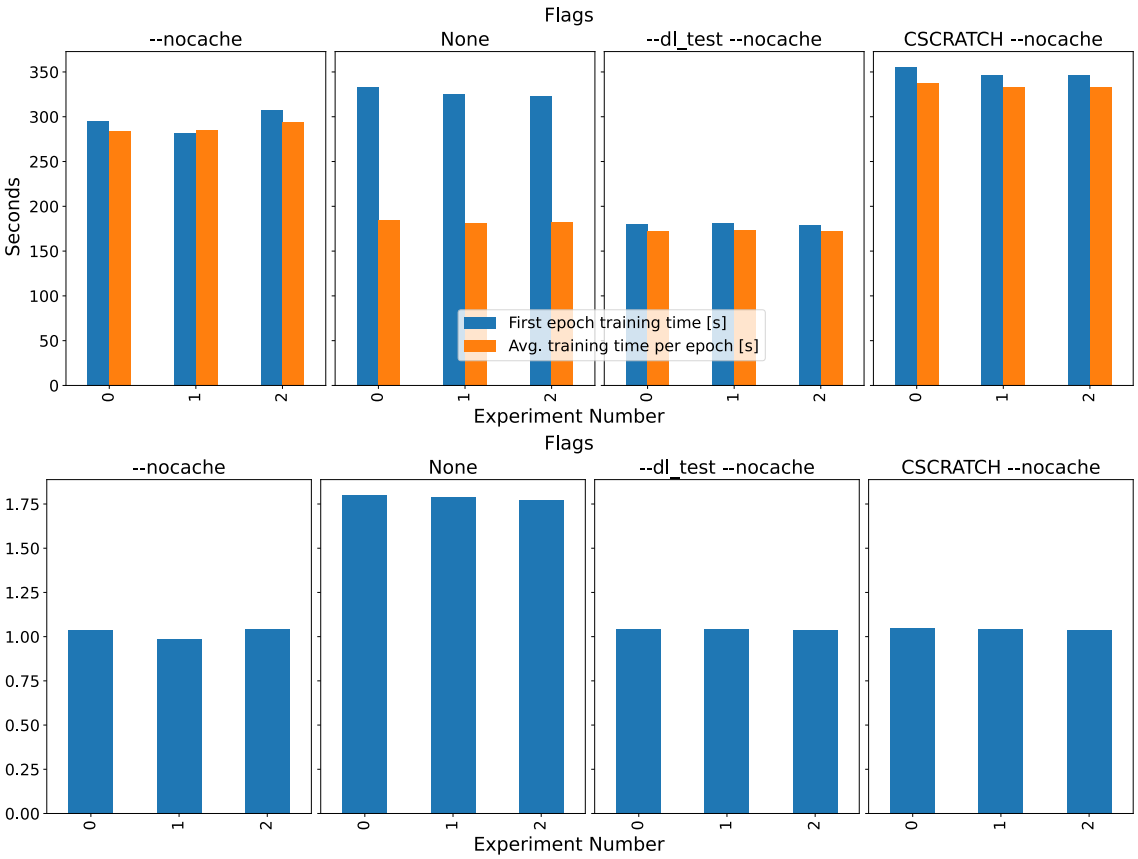



Figure 42: AP3 JUWELS Booster Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom). *ap3-jwb-epoch-time*

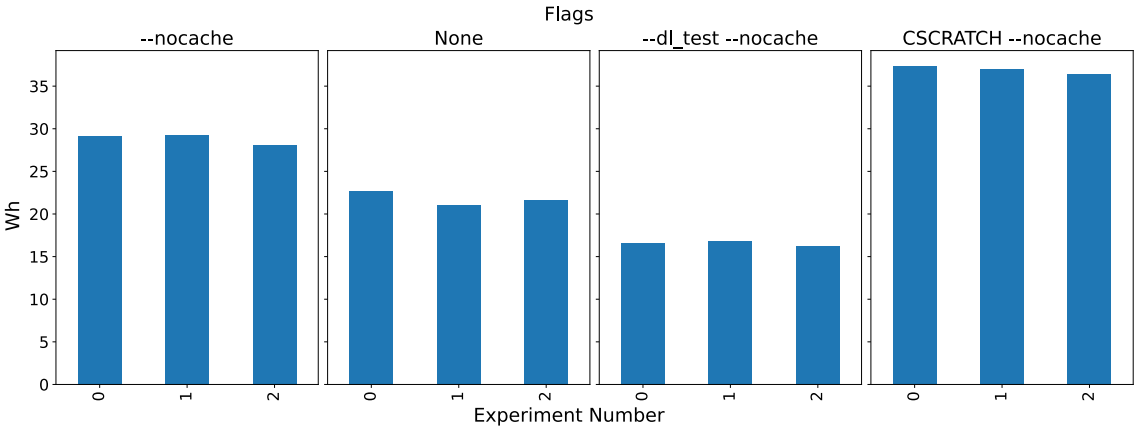



Figure 43: AP3 JUWELS Booster Energy: Total GPU energy consumption during the training phase. *ap3-jwb-energy*

the time between the end of the inference process and the end of the total runtime. Overall, we can conclude that the inference test results are very consistent and show very little variation between the three runs. The data loading overhead is also relatively small, indicating that the system is capable of loading data efficiently during inference.

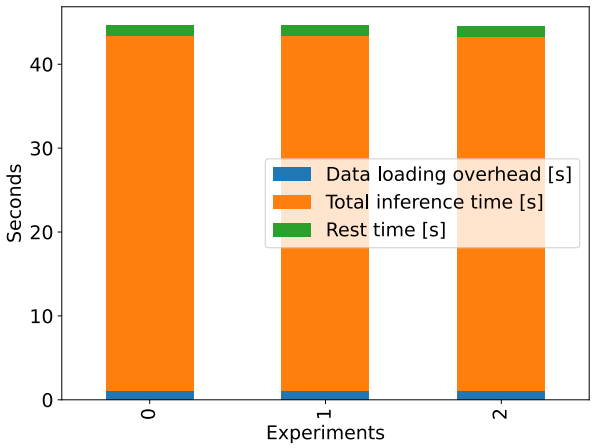
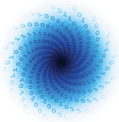



Figure 44: AP3 JUWELS Booster Inference Runtime: Runtime and relative share for multiple experiments during the inference phase *ap3-jwb-inf-runtime-share*



### 4.3.3 JUWELS Cluster

In the Juwels Cluster, 9 experiments have been conducted, divided into three triplets with different flags and configurations. The number of GPUs and MPI tasks used in these experiments was set to 1. The configuration for each triplet is as follows:

- Triplet 1: –nocache flag and SCRATCH filesystem.
- Triplet 2: no flag and SCRATCH filesystem.
- Triplet 3: –dl_test –nocache flag and SCRATCH filesystem.

The runtime of the inference phase has also been reported, with three experiments performed using this system.

#### 4.3.3.1 Runtime

The training and rest times for different runs of the AP3 using different flags but the same disk (SCRATCH) are shown in Figure Figure 45. It is worth noting that the rest times cover a very small fraction of the total runtime and are therefore negligible.
For the runs with the –nocache flag, the average total runtime is 2215.66 ± 0.08 s, while the rest time accounts for about 0.1% of the total runtime. This set of runs is the longest on Juwels Cluster, as this version avoids using TF dataset cache.
Moving to the Default version, we observe an average total runtime of 1495.64±0.08 s, which shows a decrease of approximately 32.5% with respect to the previous case.
The runs with the –dl_test flag, which test the data loading capabilities, exhibit a behavior similar to the Default case. The average runtime is 1529.37 ± 59.28 s, which represents approximately 69.0% of the training runtime in the case with the –nocache flag. The rest time in this case is 0.16% of the total runtime and is therefore negligible.
Looking at Figure 46, it is apparent that using the –nocache flag results in longer average training times per epoch compared to the default case. The average training time per epoch for the –nocache flag is higher than the default case, indicating longer training times per epoch. Furthermore, the first epoch training time for the –nocache flag is also higher than the following epochs, with a ratio ranging from 1.065 to 1.071. Overall, the use of the –nocache flag results in longer training times per epoch compared to the Default flag.
For the None configuration, the first epoch training time (mean 429.6±4.2 s) is significantly higher than the following epochs (mean 280.3±0.5 s), with a ratio

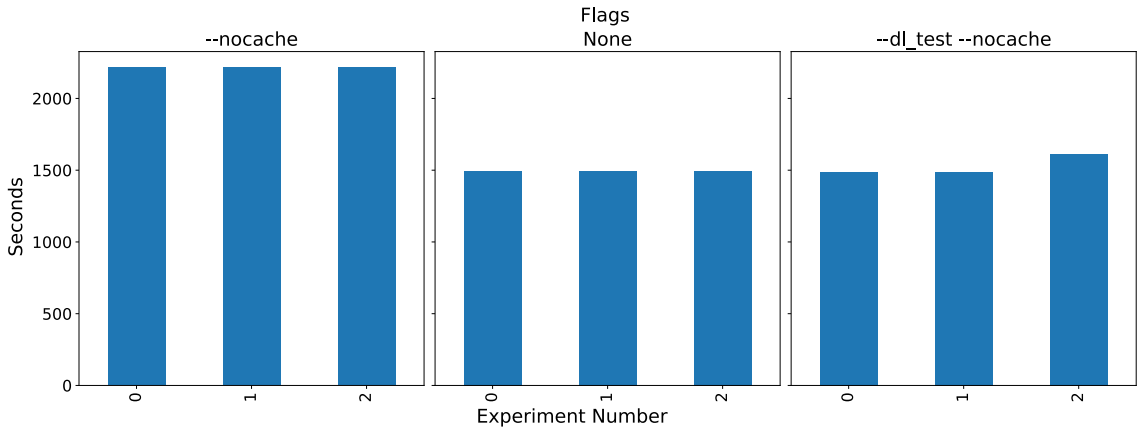



Figure 45: AP3 JUWELS Cluster Runtime: Runtime for multiple experiments during the training phase *ap3-jwc-runtime-share*

ranging from 1.519 to 1.542. This suggests that the first epoch is affected by some initial overhead.

In contrast, the average training time per epoch of the –dl_test –nocache configuration amounts to 279.33±9.70 s, which falls in the same range as the None configuration. The first epoch training time for this configuration is on average 307.33±37.90 s, and the run 0 shows a first epoch training time (351 s)which is significantly higher than the following epochs, with an average training time per epoch of 283.8 s.

It is worth noting that both the first and average epoch training time show an increase compared to JUWELS Booster, as seen in Figure Figure 46. This is in line with the increase in total training time.

On the JUWELS Cluster, the default configuration of the benchmark consumes 241.0 ± 3.5 W h of energy, as shown in Figure 47. However, when using the –nocache flag, the energy consumption increases to 347.1 ± 3.6 W h. On the other hand, when using the –dl_test –nocache flag, the average GPU energy consumption is 194.1 ± 10.1 W h, which represents the 55.9% of the energy consumed by the –nocache configuration.

#### 4.3.3.2 Inference runtime

Based on Figure 48, we can see that the total runtime of the inference test ranges from 44.24 seconds to 44.63 seconds, while the data loading overhead ranges from 1.01 seconds to 1.44 seconds.

In terms of total inference time, the experiments have similar results, with the time ranging from 42.04 seconds to 42.05 seconds. The rest time ranges from 1.15

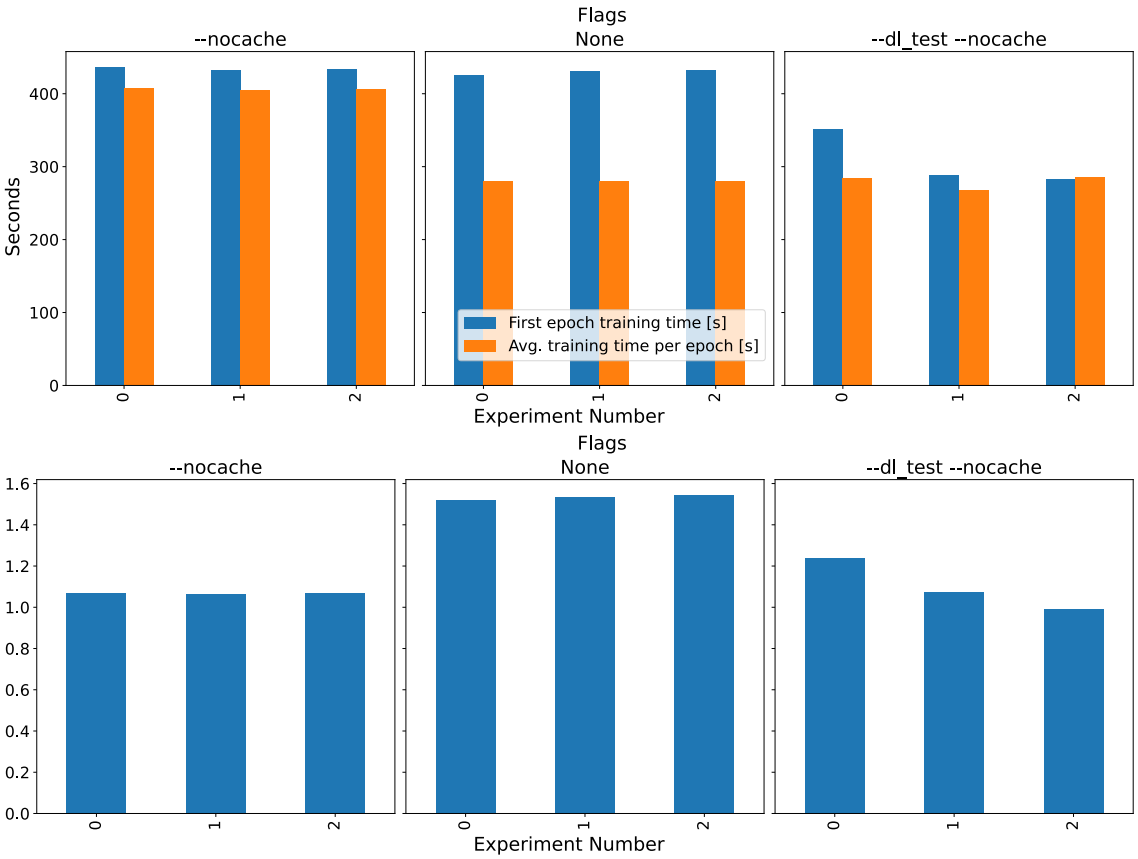



Figure 46: AP3 JUWELS Cluster Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom) *ap3-jwc-epoch-time*

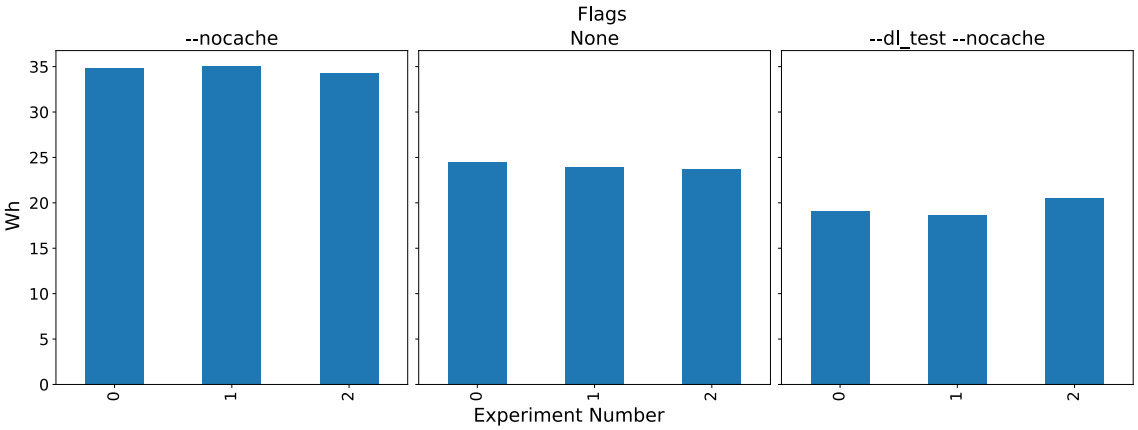



Figure 47: AP3 JUWELS Cluster Energy: Total GPU energy consumption *ap3-jwc-energy*

seconds to 1.18 seconds, with the first experiment having the shortest rest time.
When comparing these results to Juwels Booster case, we can see that the total runtime and data loading overhead times are very similar in this set of experiments, as the total inference and rest times.
Overall, these results suggest that the performance of the inference test is consistent across multiple experiments, with similar total inference times and rest times observed.

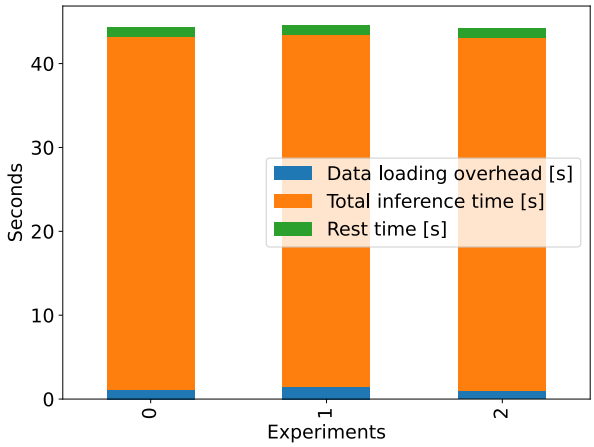
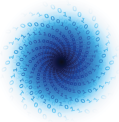



Figure 48: AP3 JUWELS Cluster Inference Runtime: Runtime and relative share for multiple experiments and two configurations *ap3-jwc-inf-runtime-share*



### 4.3.4 E4 Intel System

In the Intel System, 12 experiments have been conducted on Juwels Booster, divided into four triplets with different flags and configurations. The number of GPUs and MPI tasks used in these experiments was set to 1. The configuration for each triplet is as follows:

- Triplet 1: –nocache flag and the default (NFS) filesystem.
- Triplet 2: no flag and the default (NFS) filesystem.
- Triplet 3: –dl_test –nocache flag and the default (NFS) filesystem.
- Triplet 4: –nocache flag and NVMe filesystem.

The runtime of the inference phase has also been reported, with three experiments performed using this system.

#### 4.3.4.1 Training

As showcased in Figure 49, better training performance than JUWELS Booster can be seen on the E4 Intel System, in particular we can see that: the –nocache and NVMe –nocache configurations have similar runtimes, while the –dl_test –nocache configuration has a significantly lower runtime. None configuration has a runtime that is longer than the others.
For the –nocache flag experiments, the average total runtime is $793.93 \pm 8.10$ s, with an average total training time of $792.12 \pm 7.18$ s and a rest time of $1.81 \pm 0.10$ s, which is negligible compared to the total runtime. On the other hand, the tests with the –dl_test flag –nocache, which involve data loading, have an average total runtime of $603.39 \pm 12.16$ s, which accounts for approximately 75% of the total training time of the –nocache case.
The None configuration has the longest average total runtime of $874.17 \pm 5.71$ s.
Interestingly, the last set of runs with the –nocache flag and NVMe filesystem show an average total training time of $799.18 \pm 3.28$ s, which is consistent with the results obtained by the –nocache configuration. However, further investigation is needed to understand these results more deeply.
The epoch training time plot (Figure 50) shows that the configuration without any flags (None) takes the longest to train, with an average training time per epoch of $174.50 \pm 0.85$ seconds. This configuration reports a difference between first epoch training time and average training time per epoch of about 118.5 s.

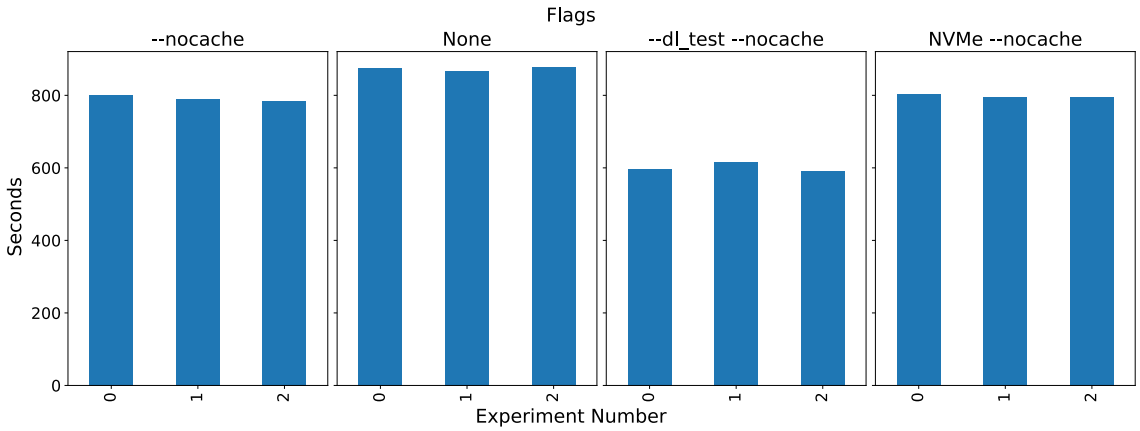



Figure 49: AP3 E4 Intel System Runtime: Runtime for multiple experiments during the training phase *ap3-e4i-runtime-share*

In contrast, the –nocache flag configuration shows an average first epoch training time of 163.30±0.58 seconds and an average training time of 158.30±0.88 seconds per epoch. The NVMe –nocache flag configuration takes slightly longer, with an average first epoch training time of 164.00 ± 0.57 s and an average training time of 159.50 ± 0.65 seconds per epoch.

As expected, the –dl_test –nocache flag configuration takes the least amount of time, with an average first epoch training time of 119.70 ± 1.33 seconds and an average training time of 120.30 ± 0.76 seconds per epoch.

Overall, the configuration with None flag takes significantly longer than the others, while the –nocache and NVMe –nocache flag configurations show similar performance.

The power consumption data for each configuration is shown in figure Figure 51, and it's evident that there is significant variability between them.

The average power consumption for the –nocache configuration is 178.29±37.30 Wh, while the None configuration has an average consumption of 161.34 ± 30.65 Wh. The –dl_test –nocache configuration has the lowest average power consumption at 104.95 ± 11.73 Wh, while the NVMe –nocache configuration has an average consumption of 155.19 ± 17.23 Wh.

It's worth noting that all configurations, except for the –dl_test –nocache configuration, have similar power consumption range.

Based on the action metric shown in Figure 52, we can observe that the configurations using –nocache and NVMe –nocache flags have similar action values, while the –dl_test –nocache configuration has significantly lower values. The configuration with no flags reports action values that are slightly longer than the others.

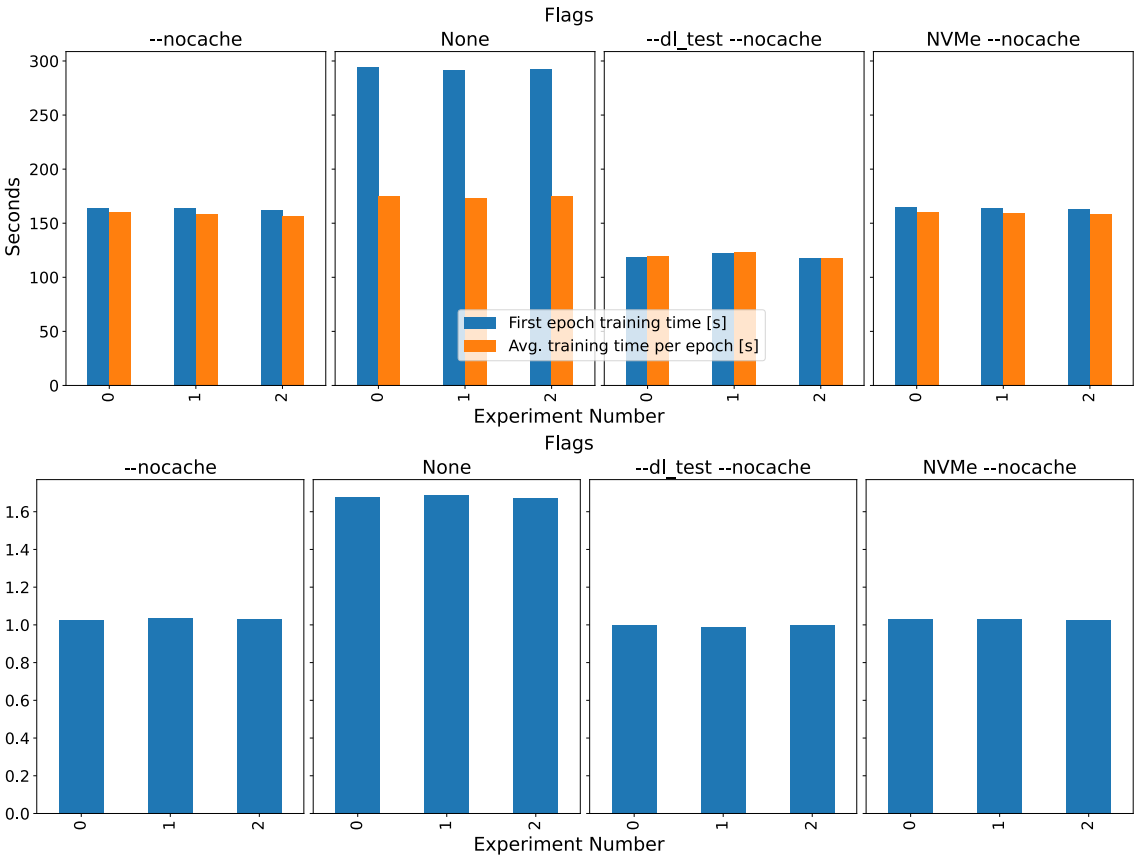



Figure 50: AP3 E4 Intel System Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom) *ap3-e4i-epoch-time*

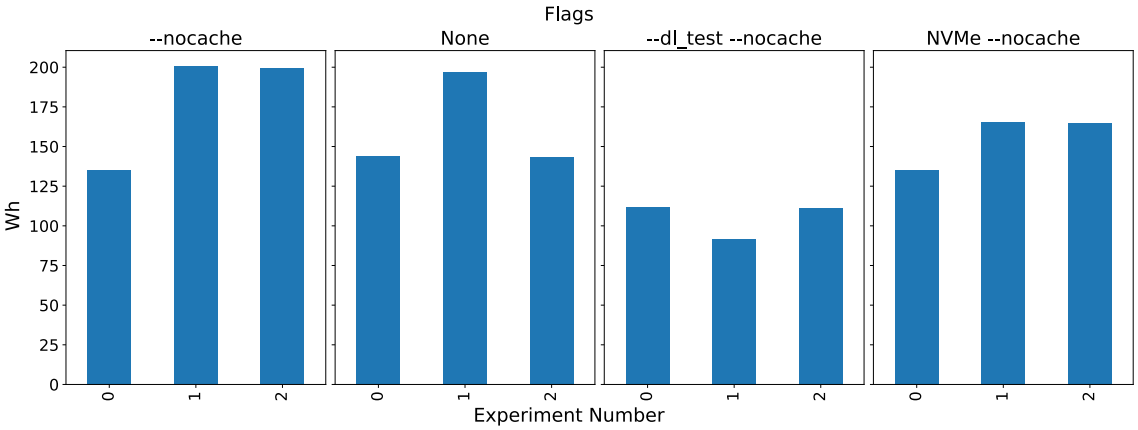



Figure 51: AP3 E4 Intel System Energy: Total node energy consumption during training phase *ap3-e4i-energy*

Looking at the action values for the different experiment configurations in Figure 52, we see that the experiments using the –nocache flag have an average action value of 508.87 ± 102.13 MJs, while the experiments with the –nocache flag and NVMe filesystem have a slightly better performance with an average action value of 446.31 ± 47.53 MJs. This indicates that the NVMe configuration is more efficient than the NFS one.
On the other hand, the tests with the –dl_test flag –nocache report an average action value of 227.60 ± 20.83 MJs, which is the smallest among all configurations, as expected. The None configuration has an average action value of 507.52 ± 92.66 MJs, which is similar to the –nocache configuration.

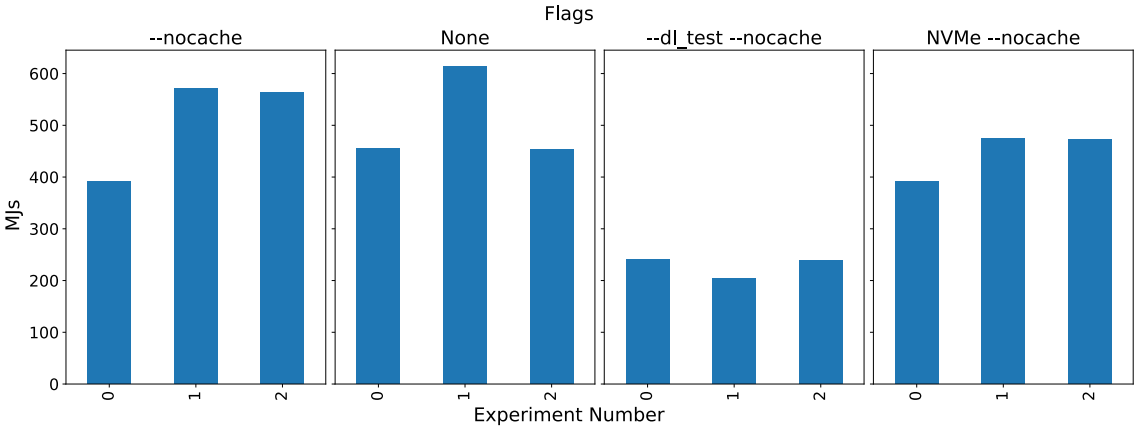



Figure 52: AP3 E4 Intel System Action: Action values (in MJs) for twelve applications using different flags during the training phase. *ap3-e4i-action*

### 4.3.4.2 Inference

Looking at Figure 53, we can see that experiment 0 had the longest total runtime (34.36 seconds), while experiment 2 had the shortest (28.41 seconds). However, experiment 0 had the longest data loading overhead (7.56 seconds), while experiment 2 had the shortest total inference time (26.91 seconds). This suggests that experiment 0 may have had more data to load into memory.
All experiments had a relatively short rest time, with the longest being only 1.18 seconds. This suggests that the inference process was relatively consistent across experiments, with little variation in the amount data loading overhead.

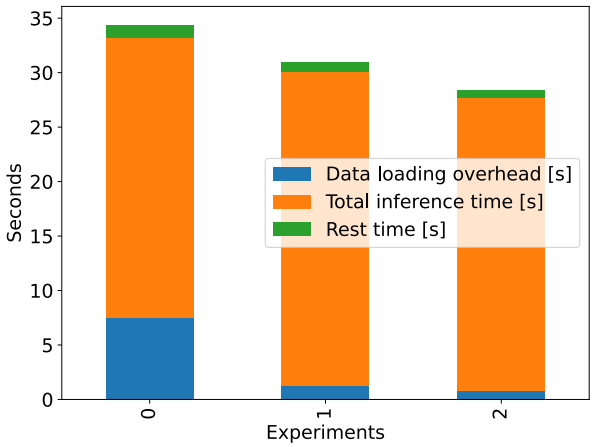



Figure 53: AP3 E4 Intel System Inference Runtime: Runtime and relative share for multiple experiments during inference phase *ap3-e4i-inf-runtime-share*

### 4.3.5 E4 AMD System

In the AMD System, 9 experiments have been conducted on Juwels Booster, divided into three triplets with different flags and configurations. The number of GPUs used in these experiments was set to 1. The configuration for each triplet is as follows:

- Triplet 1: –nocache flag and the default (NFS) filesystem.
- Triplet 2: no flag and the default (NFS) filesystem.
- Triplet 3: –dl_test –nocache flag and the default (NFS) filesystem.

The runtime of the inference phase has also been reported, with three experiments performed using this system.

#### 4.3.5.1 Training

In Figure 54 we plotted the total training time, which cover the 99.8% of the total runtime, for the nine experiment performed on the AMD System.
The three experiment performed with the –nocache flag are the one with the greatest runtime that amounts to $2377.7 \pm 290.9$ seconds. Then follows the experiments without flag, with an average total training time of $2025.6 \pm 170.8$ seconds. For what concerns the data loading instead, AMD show the best performance between the four provided systems, with an average runtime of $235.1 \pm 1.3$ seconds.

Even if this system is very fast in the data loading process, it results to be the slowest in the training process.
In Figure 54, we can observe the total training time of nine experiments performed on the AMD System, which covers 99.8% of the total runtime, while the 0.2% is spent in the so called rest time which is negligible. Among these experiments, those performed with the –nocache flag have the longest training runtime, with an average of $2377.7 \pm 290.9$ seconds. Also, experiments without the flag have a lower average total training time of $2025.6 \pm 170.8$ seconds.
In terms of data loading, the AMD System shows the best performance among the four systems, with an average runtime of $235.1 \pm 1.3$ seconds. However, despite the quick data loading, the AMD System is the slowest among the four systems when it comes to training process.

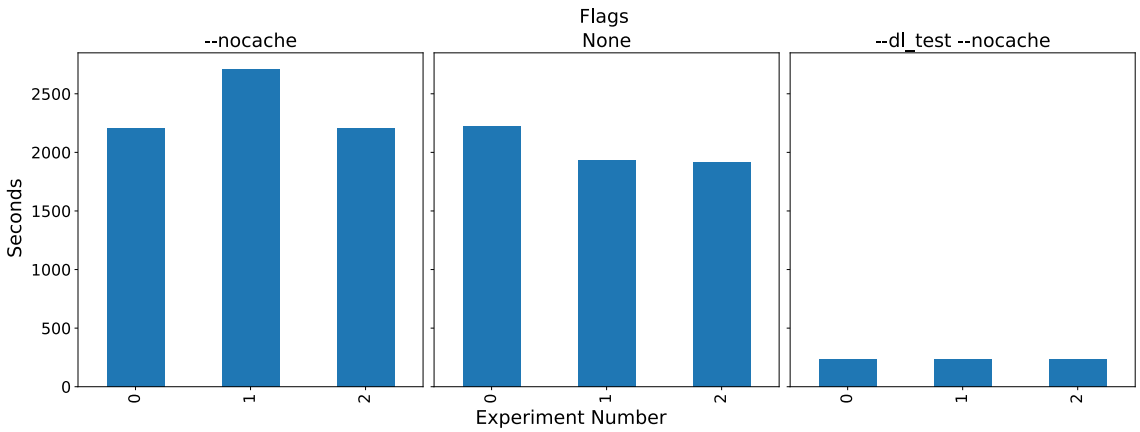



Figure 54: AP3 E4 AMD System Runtime: Runtime for multiple experiments during the training phase *ap3-e4amd-runtime-share*

In analyzing the comparison between epoch training time, we observe that the –nocache flag exhibits a significant difference between the first epoch training time and the average training time. This is unexpected as we would normally expect this difference to be small in the nocache configuration. The average difference between the first and average epoch training time is $385.50 \pm 227.71$ seconds.
To contrast with the previous observation, the experiments conducted without flags exhibit a smaller difference between the first and average epoch training time, varying from 5.2 seconds (run 2) to 250.6 seconds (run 0), which is a considerable range. Notably, for experiments 1 and 2, this difference is only 6 and 5.2 seconds, respectively, which is an unexpected behavior for the None configuration (it would be more reasonable for the no-cache case).
As for the data loading test, we see, as expected, a very small difference between

the first and average epoch training time ($0.6 \pm 0.2$ seconds) and an average training time of 47 seconds per epoch.
Further investigation is necessary to understand the behavior observed with the –nocache and no flag configurations on the AMD System.

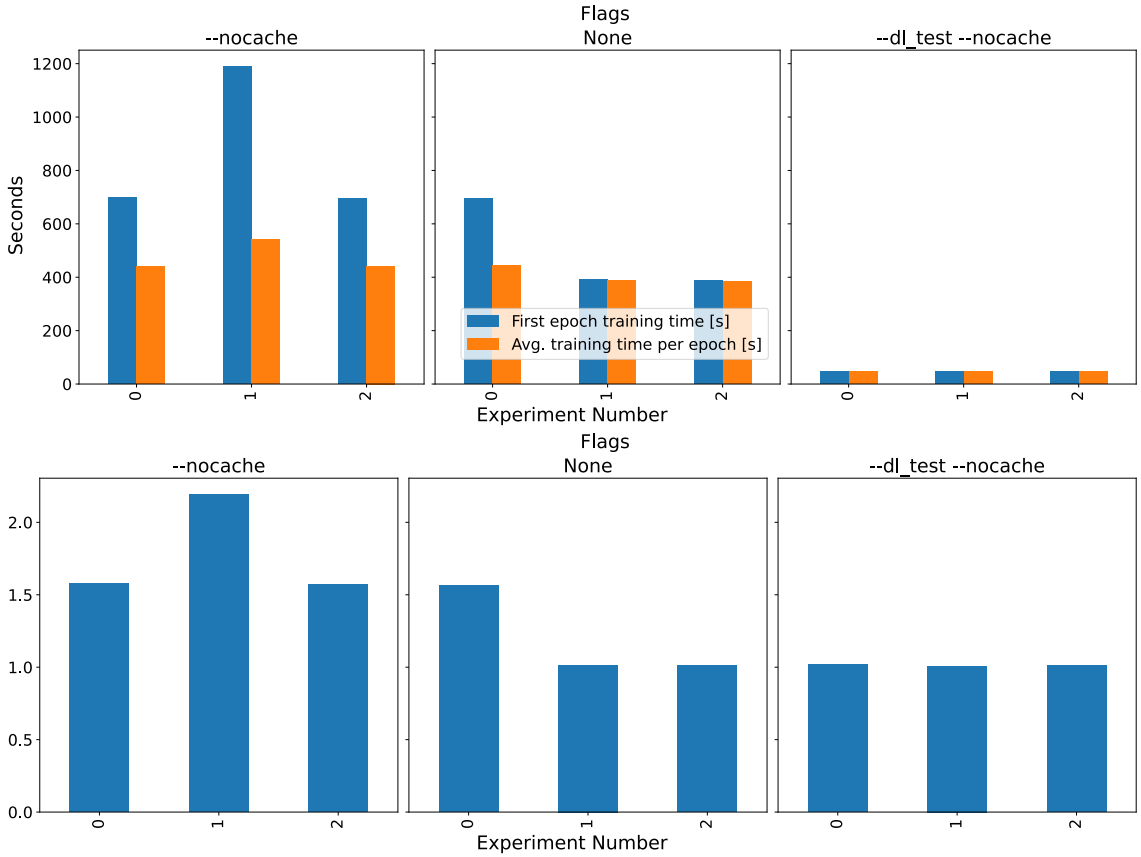



Figure 55: AP3 E4 AMD System Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom) *ap3-e4amd-epoch-time*

In the figure provided, Figure 56, we can observe that the node energy consumption is generally higher for experiments that were performed with the –nocache flag compared to those without the flag. For instance, the average energy consumption for experiments 0, 1, and 2 with the –nocache flag is $377.9 \pm 9.2$ Wh, whereas it is $313.4 \pm 10.1$ Wh for experiments 0, 1, and 2 without the flag. This comparison is consistent with our earlier analysis of the application's runtime.
Moreover, we can see that the energy consumption for the data loading test is significantly lower than that of the training experiments, as expected, since data loading is a less computationally intensive task on the AMD system. The average energy consumption for experiments with the –dl_test –nocache flag is $39.8 \pm 0.3$ Wh.

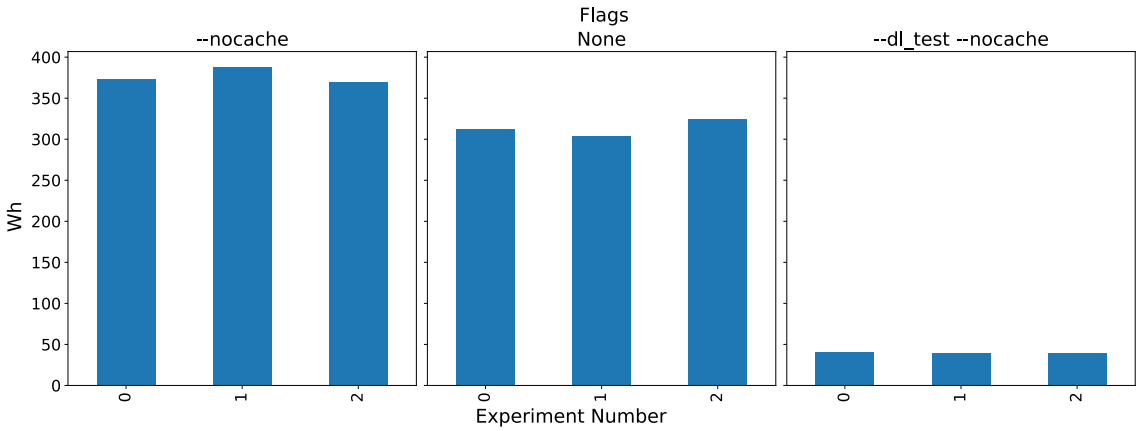



Figure 56: AP3 E4 AMD System Energy: Total node energy consumption during the training phase *ap3-e4a-energy*

As shown in Figure 57, in the set of experiments using the –nocache flag, the average action value is $3239.05 \pm 476.68$ MJs, which is the highest among all configurations. The experiments without any flag have an average action value of $2287.65 \pm 197.58$ MJs.
In terms of data loading performance, the AMD system outperforms the E4 system, with an average runtime of $33.69 \pm 0.44$ seconds. Interestingly, the experiments with no flags show the best performance for data loading in this set of experiments.

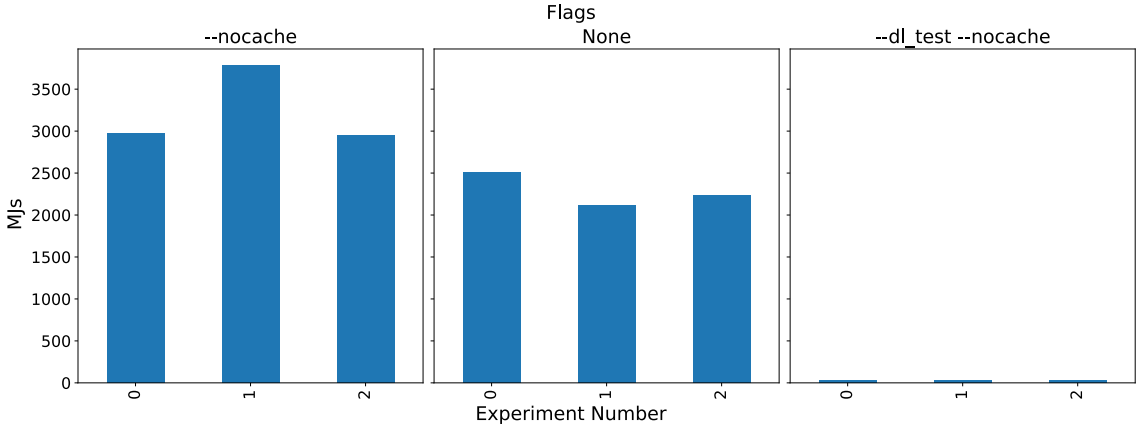



Figure 57: AP3 E4 AMD System Action: Action values (in MJs) for nine applications using different flags during the training phase. *ap3-e4a-action*

### 4.3.5.2 Inference

For the inference case, three different experiment were performed. As can be seen in Figure 58, the total runtime of the inference is shared between the data loading

overhead, the actual inference time, and the rest time. The data loading overhead is quite small, ranging from 0.61 to 0.66 seconds, while the actual inference time ranges from 13.15 to 13.69 seconds. The rest time is consistent across all experiments, at approximately 1.4 seconds. In conclusion, we can say that the total runtime of the inference test is primarily determined by the actual inference time, with data loading overhead and rest time being relatively small components of the total runtime.

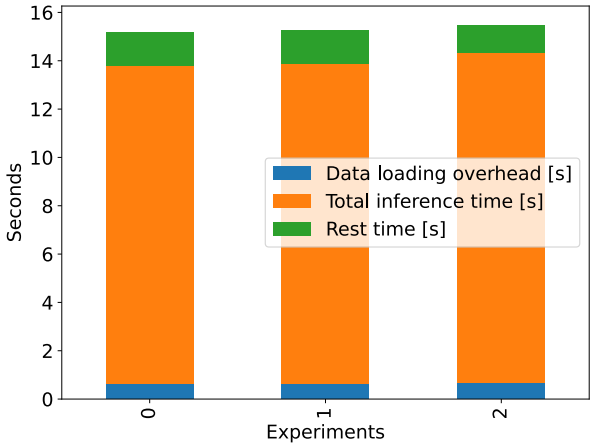
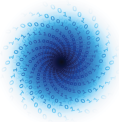
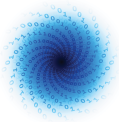



Figure 58: AP3 E4 AMD System Inference Runtime: Runtime and relative share for multiple experiments and two configurations *ap3-e4a-inf-runtime-share*



### 4.3.6 Results

The measurements conducted in the AP3 context indicate that most of the runtime (about 99 %) is spent in training. However, tests performed with the –dl_test flag show that the application streams in data while training and there is strong evidence of the performance being I/O-bound, except for the AMD system, which reports a much lower runtime than the others in the data loading test.
Generally, the training time decreases from the AMD system (which reports the longest runtime) with the Mi100 GPUs, to the Intel system, which reports the shortest runtimes with the A100 GPU. In terms of inference, the best performance in terms of time is obtained by the AMD system, with an average of 15.3 seconds, while the longest runtimes were recorded in the JSC systems, which report very similar values and average 44.4 seconds.
In terms of power consumption, the power consumption of individual GPUs in JSC systems is slightly higher in the Cluster case (about 16% higher than in the Booster case). In the E4 systems, where we have the measurements inherent to the consumption of the entire node, we can see that the consumption related to the flags of the training phases is halved in the Intel system, while the AMD system reports the lowest consumption for the test runs on data loading.
Considering the training tests with the –nocache flag and the None case, which are common to both the Intel and AMD systems, it can be seen that the Intel system outperforms the AMD system. The average action value between the –nocache and None configuration is, for the Intel system, $508.19 \pm 87.22$ MJs, while the AMD system has a much higher value of $2763.35 \pm 614.86$ MJs.
On the other hand, when it comes to data-loading tests, the AMD system performs better with an average action value of $33.69 \pm 0.44$ MJs, compared to the $227.60 \pm 20.83$ MJs of the Intel system, as also reported in the analysis.
Regarding the use of different filesystems, using HPST and NVMe did not provide the expected performance improvement in terms of runtime and consumption. This aspect will be investigated more closely to achieve the expected performance for the next deliverable.



## 4.4 AP 4

### 4.4.1 Notes

| Training dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 64 GB | 2.5 GB | 1889 | [14, 361, 720] | 1 |

| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| 633698 | 0 | CRPS | None |

| Data formats | Frameworks (to be) used |
|---|---|
| NetCDF (.nc) | PyTorch 1.11 |

Application 4 aims to improve the precision and effectiveness of weather forecasts by utilizing deep neural networks to process the ensemble outputs of numerical weather prediction systems. This is achieved through the use of the ENS-10[3] dataset, which contains ten ensemble members spanning 20 years from 1998 to 2017. The UNet model is used to predict geopotential at 500 hPa, represented by Z500.
To train the model, we used the entire ENS-10 dataset at 500hPa and used the ERA-5 dataset as the ground truth. The model was trained for three epochs with a batch size of one, and the Adam optimizer was used in all our experiments. In addition, we utilized the NetCDF data format and implemented a PyTorch dataloader to efficiently process the data for the model.
The underlying data of illustrations in this section can be found in appendix 6.4.

#### 4.4.1.1 IO Issue

To run our application, we implement a PyTorch dataset. To this end, we use the xarray package to read the NetCDF files. We found that this is the main bottleneck in our IO as for every data point, we extract the corresponding date in our dataset from the NetCDF file, calculate the mean and standard deviation over all ensembles, and form the PyTorch tensor. We repeat this process for every single access to

[3] https://arxiv.org/abs/2206.14786

the dataset, which takes much time. We plan to fix this issue by preprocessing all the data points before training time and saving them into other formats (like .NPY), or using a more efficient data loader. We investigate this issue in the following steps.

### 4.4.2 JUWELS Booster

In the Juwels Booster, within the AP4, a total of 6 runs were performed. These runs were organized into two groups, with each group consisting of 3 runs. The difference between the two groups was the filesystem used during the experiments. In the first group of 3 runs, the GPFS filesystem was utilized, while in the second group, the HPST was used.

#### 4.4.2.1 Training

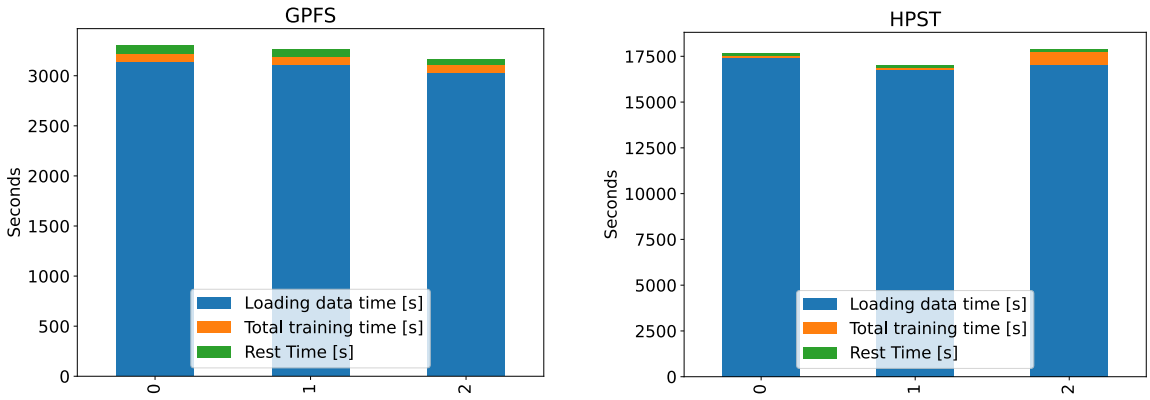



Figure 59: AP4 JUWELS Booster Runtime: Runtime and relative share for multiple experiments *ap4-jwb-runtime-share*

To perform these tests, we use a single A100 GPU on the Juwels Booster for each experiment and repeat them three times with different random seeds. Some variation between different benchmark runs can be observed in Figure 59 with regard to the distribution of time spent between loading data, training time, and rest time, as well as the absolute time spent on training. It shows that using the current data format, almost all the runtime is spent on the IO. On the GPFS (SCRATCH directory), loading data time varies between 95 % and 96 % of the total runtime.
As the application is IO-bounded, the total runtime is affecected by changing the file system. To this end, on the HPST (CSCRATCH directory), the loading data time

varies between 94.9 % and 98.7 %, while training time between varies between 0.4 % and 4.3 %.

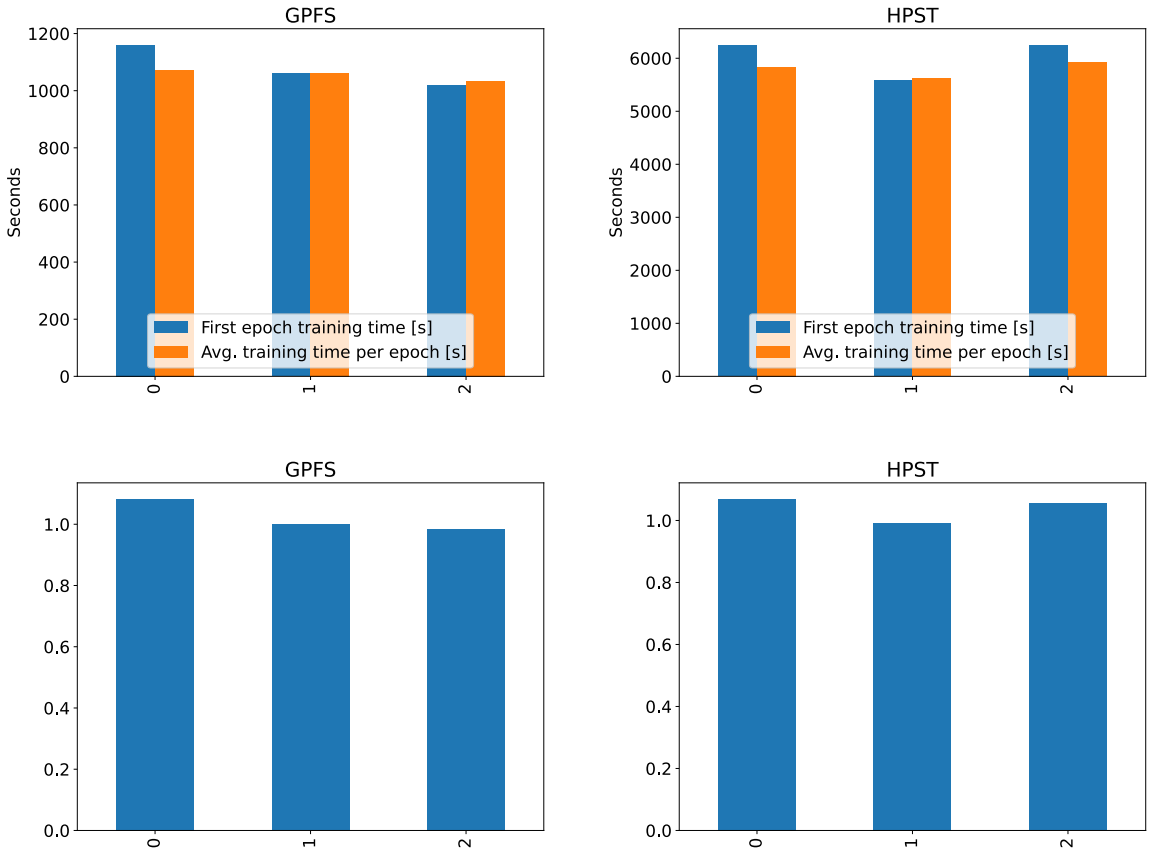



Figure 60: AP4 JUWELS Booster Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom) *ap4-jwb-epoch-time*

The time to train the first training epoch and the average of all epochs is shown in Figure 60 for different repeating runs. The first epoch takes more time in almost all of the runs, possibly due to the compilation overhead for building parts of the computational graph for training. In addition, as the application is IO-bounded, we see that training the model in the HPST file system (CSCRATCH directory) takes more runtime (almost 5.5 times more).
Due to the large difference in runtime, Figure 61 shows that there is a substantial difference also in the energy consumption of the GPU when using the CSCRATCH and SCRATCH disks. On average, the GPU energy consumption is significantly higher when using the CSCRATCH disk, with an average of $285.26 \pm 5.35$ Wh compared to

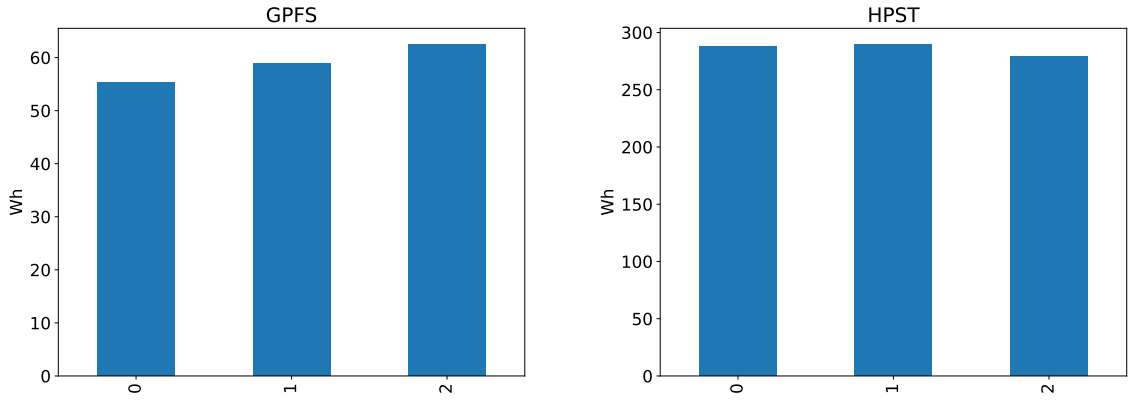



Figure 61: AP4 JUWELS Booster Energy: Total GPU energy consumption for the training phase *ap4-jwb-energy*

an average of $58.9 \pm 3.5$ Wh when using the SCRATCH disk.

### 4.4.3 JUWELS Cluster

Also in the Juwels Cluster, a total of 6 runs were performed. These runs were organized into two groups, with each group consisting of 3 runs. The distinction between the two groups was the filesystem used during the experiments.
In the first group of 3 runs, the GPFS filesystem was utilized, while the HPST was used in the second group.

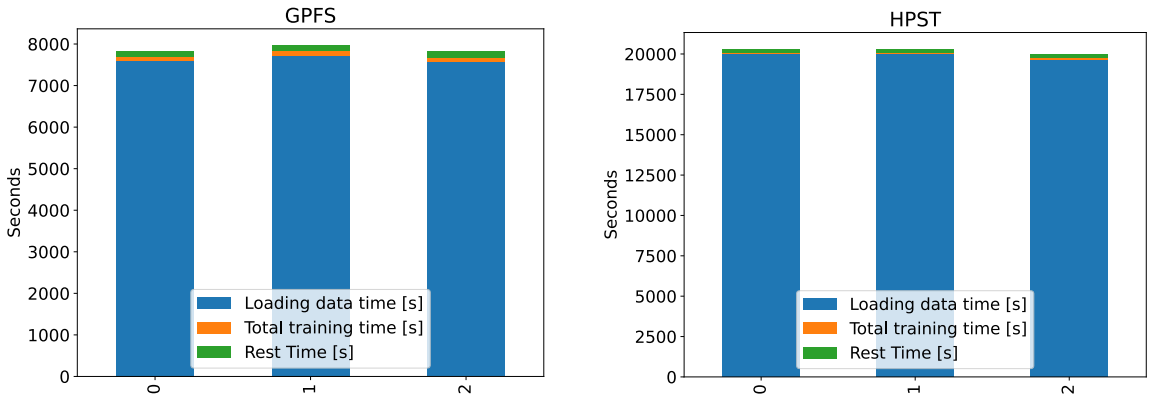
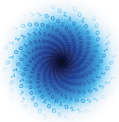



Figure 62: AP4 JUWELS Cluster Runtime: Runtime and relative share for multiple experiments *ap4-jwc-runtime-share*

On Juwels Cluster we use a single V100 GPU on the JUWELS cluster for each experiment and repeat them three times with different random seeds (similar to the



JUWELS BOOSTER experiments). Some variation between different benchmark runs can be observed in Figure 62 with regard to the distribution of time spent between loading data, training time, and rest time, as well as the absolute time spent on training. Again, our results show that most of the runtime is spent on loading the data. This may be because of the overhead of processing large NetCDF formats and generating the PyTorch tensor from those files. On the GPFS (SCRATCH directory), loading data time is about 97.1 %, while training time varies between 1.0 % and 1.2 %. On the HPST (CSCRATCH directory), the loading data time is about 98.4 % of the total runtime, while training time varies between 0.3 % and 0.4 %.
The time to train the first training epoch and the average of all epochs is shown in Figure 63 for different repeating runs. In most of the runs, the first epoch takes slightly longer, which could be attributed to the compilation overhead of building parts of the computational graph for training. However, this difference is not significant when compared to the time taken for the other epochs. Interestingly, we observe that training the model in the HPST file system (CSCRATCH directory) takes significantly longer to execute, taking approximately 2.6 times more compared to the GPFS case.

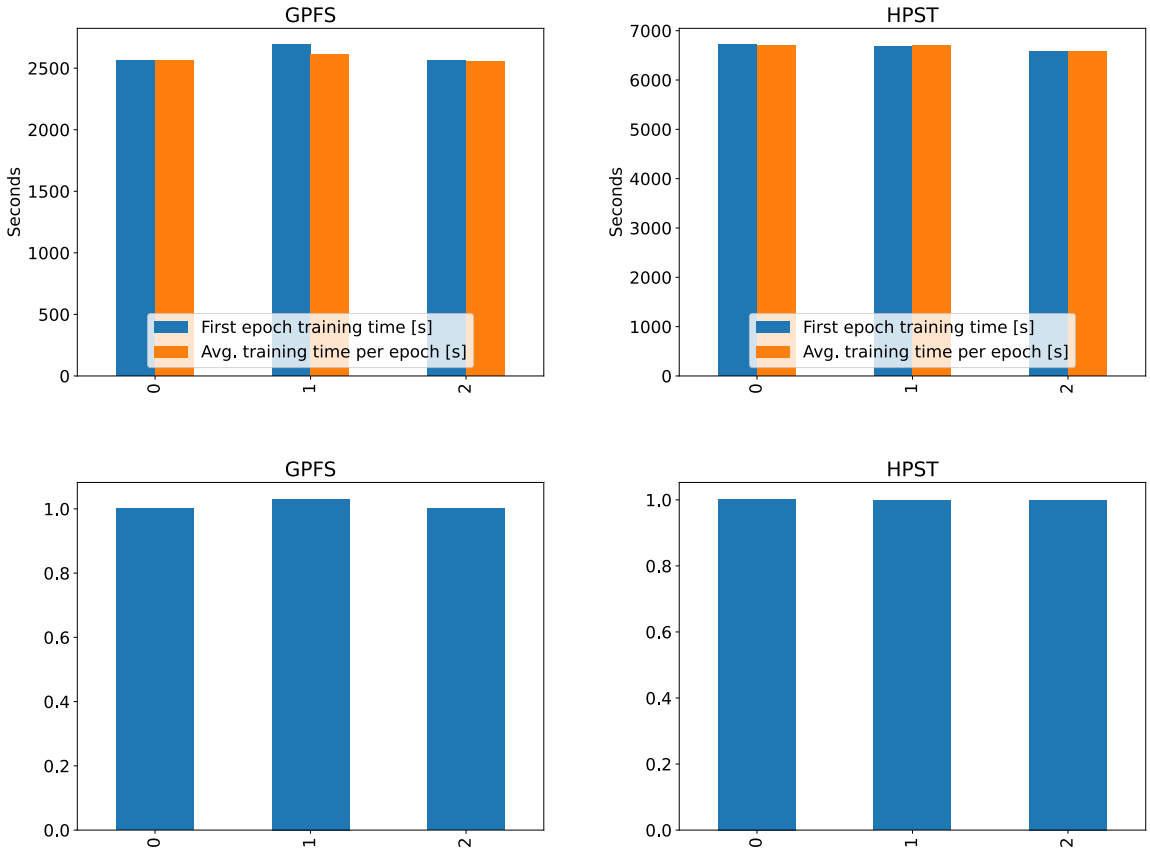



Figure 63: AP4 JUWELS Cluster Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom) *ap4-jwc-epoch-time*

### 4.4.4 E4 Intel System

Within the AP4, only the Intel System provided by E4 was used, and with this system three runs have been performed.
Each experiment was executed on a single node with an A100 GPU and the NFS filesystem, the obtained results in term of runtime are represented in Figure 64, where some variation between different benchmark runs can be observed with regard to the distribution of time spent between loading data, training time, and rest time. As in the Juwels cases, using the current data format almost all the runtime is spent on the IO. In fact we can see that loading data time is the largest component of the total runtime for each run, which covers 96.8 % and 97.4 % of the total runtime, suggesting that it is the significant bottleneck in the training process of the AP4.

The time spent in training varies between 0.7 % and 0.9 % and it is roughly the same for each run. Then there is the rest time (time is spent on tasks such as saving checkpoints or updating the model's parameters) that varies between 1.7 % and 2.2 % of the total runtime.

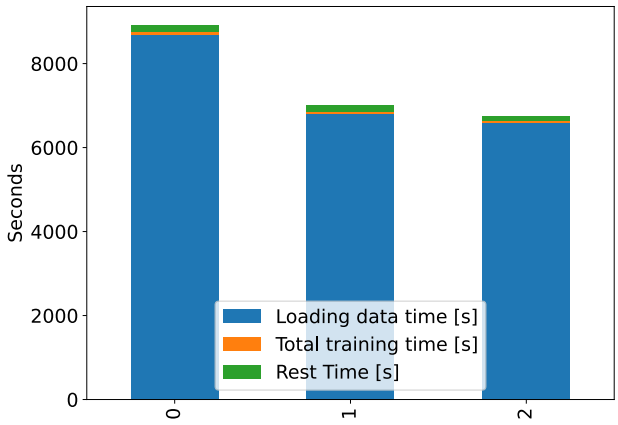



Figure 64: AP4 E4 Intel System Epoch Time: Comparison of time for first epoch and average time for an epoch (left); ratio of both quantities (right) *ap4-e4i-runtime*

Figure 63 illustrates the comparison between the average training time per epoch and the time to train the first epoch for different runs. It is worth noting that only in the first run, the first epoch takes slightly longer, which could be due to the overhead of building parts of the computational graph for training. However, this difference is not significant for runs 2 and 3, as the ratio between the time to train the first epoch and the average time per epoch is almost 1.

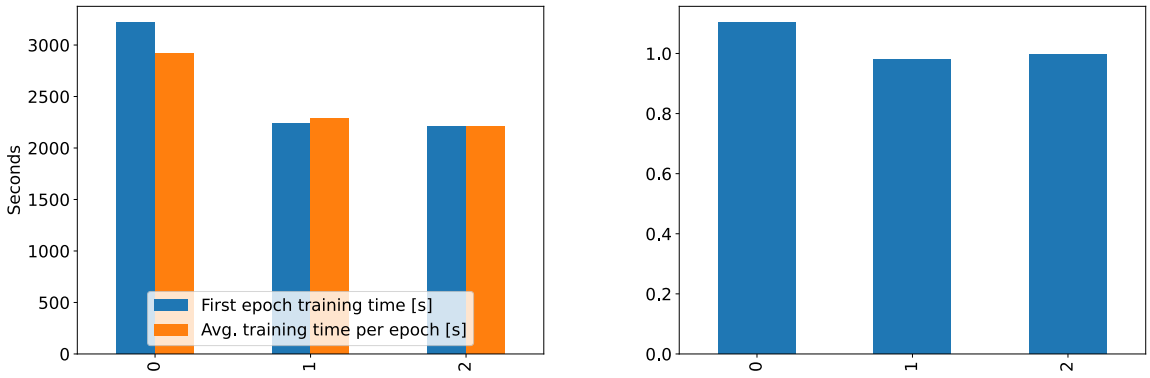



Figure 65: AP4 E4 Intel System Epoch Time: Comparison of time for first epoch and average time for an epoch (left); ratio of both quantities (right) *ap4-e4i-epoch-time*

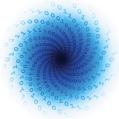
### 4.4.5 Results

Based on the analysis and the information provided at the beginning of the section, the data loading part is a significant bottleneck for this application, accounting for between 95 % to 97 % of the total runtime.

In the AP4 benchmarks, the longest runtimes were observed, taking 3158 s to 3305 s on JUWELS Booster and 7814 s to 7967 s on the JUWELS cluster with GPFS filesystem, while on the Intel system and NFS filesystem it took 6754 s to 8914 s. However, when using the HPST disk, the runtime on the JSC systems increases significantly, taking between 17 010 s to 17 914 s on Juwels Booster and 19 971 s to 20 314 s on Juwels Cluster.

This is due to the large dataset size (ENS-10 dataset is 3TB of data) used by AP4. However, despite the ease of use of the NetCDF format, experiments have shown that processing the data using this format adds considerable overhead. In addition, AP4 experiences an issue with the HPST disk, as mentioned in this document, where the requirement of FUSE Direct-IO on the HPST causes some single process access with specific patterns to be slower compared to directly accessing SCRATCH. AP4 appears to suffer the most from this performance drop when using the HPST. We are currently working with the system administrators to further investigate this performance drop and methods to possibly mitigate it.

Due to technical problems, we were only able to report GPU consumption in the Booster case, which, in the HPST case, is five times higher than the consumption in the GPFS case.



## 4.5 AP 5

### 4.5.1 Notes

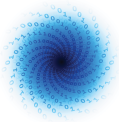
| Data formats | Frameworks (to be) used |
|---|---|
| NetCDF | Tensorflow v2.6.0 with Keras API |

The raw data associated to graphs of this section is listed in appendix 6.5.

| Training dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 47.43 GB | 4.2 GB | 94 052 | [96, 120, 10] | 32/192 |

| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| 5 113 819 | 5280 | Earth Mover distance, gradient penalty and L1 | The benchmarked WGAN performs adversarial optimization of its generator and critic model. The critic gets updated 5x before the gnerator gets updated once (6 substeps). While the batch-size for each substep is 32, the total batch-size for iteration is 192=6x32. |

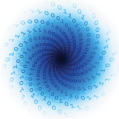
In scope of this benchmarking effort, the same dataset as in deliverable 1.3 is used. This, so-called Tier-2, dataset provides 13 years of paired ERA5- and COSMO-REA6 data where the former comprises several coarse-grained predictors for the target downscaling product, the high-resolved 2m temperature field of the latter reanalysis dataset. With a target domain of 120 × 96 grid points in longitude and latitude direction, eleven variables and more than 94K samples, the training dataset requires 47.73 GB of memory. Even though this dataset fits into the CPU memory of all benchmarked HPC-systems, a new data loader has been developed for this deliverable. The motivation for this is to enable future usage of datasets which are too large to fit into memory.
In short, the data loader is built up on a 3 stage processing pipeline. In the first step, a randomly sampled subset of monthly netCDF-file is read into memory. Note that this constitutes a notable difference to the data loader in AP 1, where only one netCDF-file is processed. Splitting up the complete dataset (comprising 132 netCDF-files) into subsets of monthly netCDF-files ensures that proper Monte Carlo



sampling along the time-dimension is performed, while the data still fits into memory. Here, one subset comprises 33 files. During the reading process, variables of interest are extracted and normalized. In the subsequent processing steps, further shuffling (Monte Carlo sampling) is performed on the samples of the data subset and the data is split up into input and target data for the supervised optimization of the downscaling neural network.

As benchmark neural network, we choose the Wasserstein Generative Adeversarial Network (WGAN) of deliverable 1.3. This WGAN deploys an U-Net as generator, whereas a conventional convolutional network serves as critic model. Due to the fact that the critic model is updated more often than the generator, one training step consumes several mini-batches, in particular six in our experiments. Thus, the data pipeline during training (implemented with Tensorflow's `tf.data.Dataset` API) provides $192 = 6 \times 32$ samples per training step.

The WGAN is trained on a single GPU only, whereas future work is devoted to enable data-distributed training.



### 4.5.2 JUWELS Booster

Within AP5, a total of six runs were conducted to evaluate the training and inference performance on Juwels Booster. Three of these runs used the GPFS filesystem located in SCRATCH, while the other three used the HPST filesystem located in CSCRATCH. All runs were conducted under the same configuration to enable a comparison of the impact of the two filesystems both on training and inference.

#### 4.5.2.1 Training

In Figure 66 we report the distribution of time spent between loading data and training time for each experiments. It is important to notice that the data loading and training processes occur simultaneously. The CPU is responsible for reading and processing data from the disk, including prefetching, and at the same time, it supplies the GPU with the data to perform the forward and backward steps of the neural network. This way, the GPU is continuously engaged in training the network while the CPU is continuously preparing a new mini-batch for the upcoming iteration step. This allows for efficient usage of both the CPU and GPU resources and results in faster training times.
In the GPFS case, the average total runtime is $2909.1 \pm 25.7$, with the training process taking up $2882.1 \pm 24.6$ on average and data loading requiring $1949.0 \pm 10.2$ On the other hand, the HPST case has a total runtime of $2829.2 \pm 14.4$, with a mean training time and data loading time of $2785.7 \pm 17.6$ and $2374.2 \pm 48.5$ s, respectively.
Overall, the HPST case is faster than the GPFS case in terms of total runtime and training time. However, data loading takes longer in the HPST case compared to the GPFS case.
In Figure 67, we can observe the time taken to train the first epoch and the average training time per epoch for multiple runs, in both GPFS and HPST file systems. Across all runs, the first epoch takes slightly longer compared to the average training time per epoch. Specifically, in the GPFS case, the first epoch training time is $609.0 \pm 8.2$ s on average, while the average training time per epoch is $576.4 \pm 4.9$ s. For the HPST case, these values are $593.0 \pm 3.6$ s and $557.1 \pm 3.5$ s, respectively. The time difference between the first epoch training time and the average training time per epoch is almost the same for both GPFS and HPST, amounting to $34.2 \pm 4.3$ s. However, the average training time per epoch is slightly lower in the HPST case, indicating that the HPST file system is faster in epoch training compared to the GPFS file system.

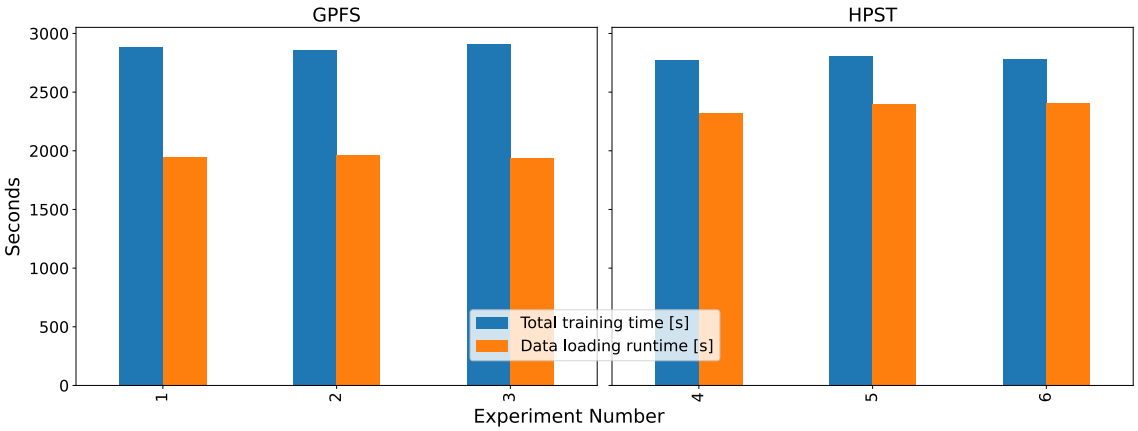



Figure 66: AP5 JUWELS Booster Runtime: Runtime and relative share for multiple experiments during the training phase *ap5-jwb-runtime-share*

The figure Figure 68 shows that the energy consumption related to the GPU is consistent across the different disks (SCRATCH and CSCRATCH), ranging from 79.2 Wh to 86.7 Wh. On average, the GPU consumption is about $83.4 \pm 3.1$ Wh.

#### 4.5.2.2 Inference

Six experiments have been performed using both SCRATCH and CSCRATCH to test the inference and the results are shown in Figure 69. Looking at the total inference time, we can see that the times range from 9.96 s to 17.87 s for GPFS with an average of $12.39 \pm 3.22$ s, and from 9.12 s to 16.48 s for CSCRATCH with an average of $12.21 \pm 3.06$ s.
Breaking down the components further, we can see that for GPFS, the model loading times range from 2.87 s to 3.85 s with an average of $3.21 \pm 0.42$ s, and the data loading times range from 9.2 s to 9.41 s with an average of $9.29 \pm 0.10$ s. For CSCRATCH, the model loading times range from 2.713 s to 3.811 s with an average of $3.18 \pm 0.44$ s, and the data loading times range from 18.78 s to 20.02 s with an average of $19.18 \pm 0.56$ s.
Overall, the model loading component is uniform across all experiments, and the same is true for the total inference time. The largest difference between the two filesystems is found in data loading, which takes just over twice as long in the CSCRATCH case compared to the SCRATCH case.

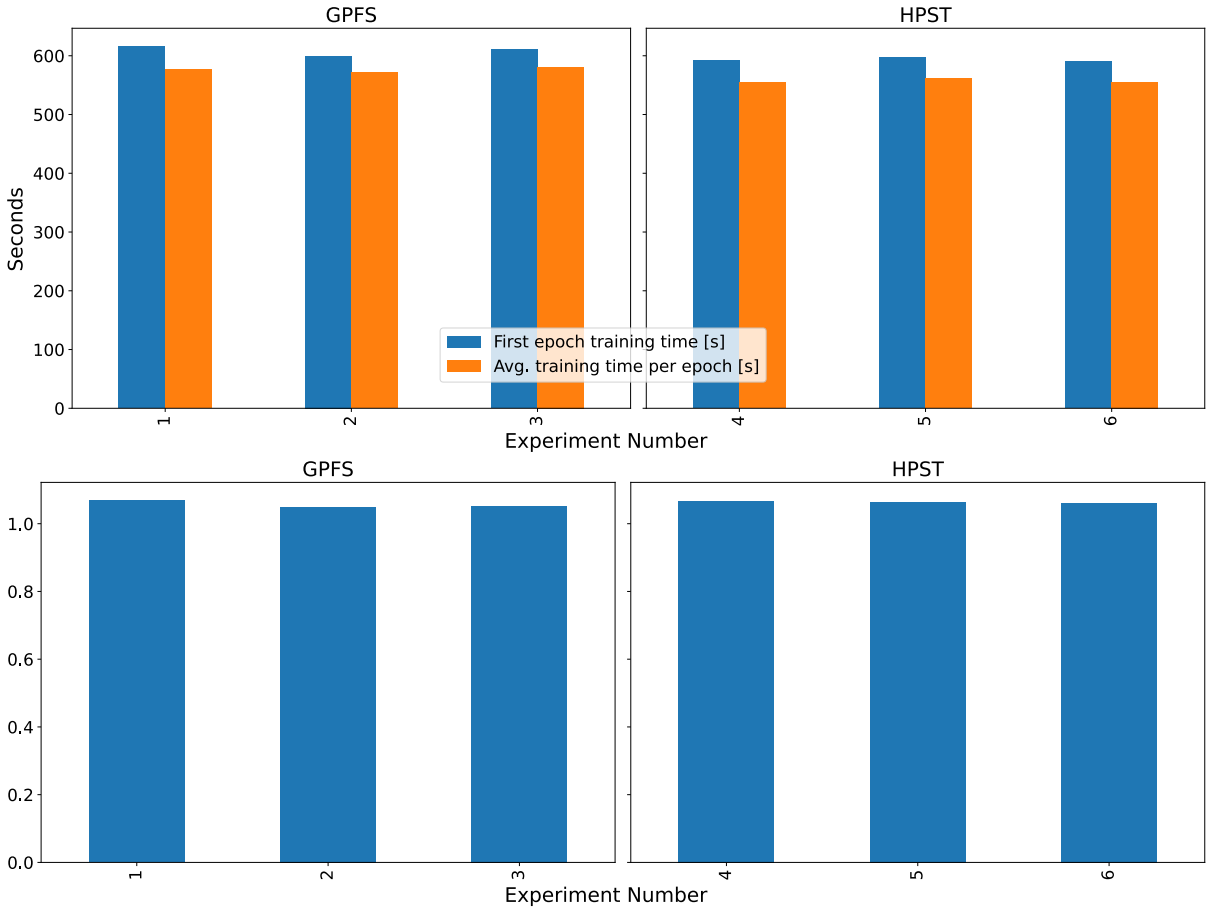



Figure 67: AP5 JUWELS Booster Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom) *ap5-jwb-epoch-time*

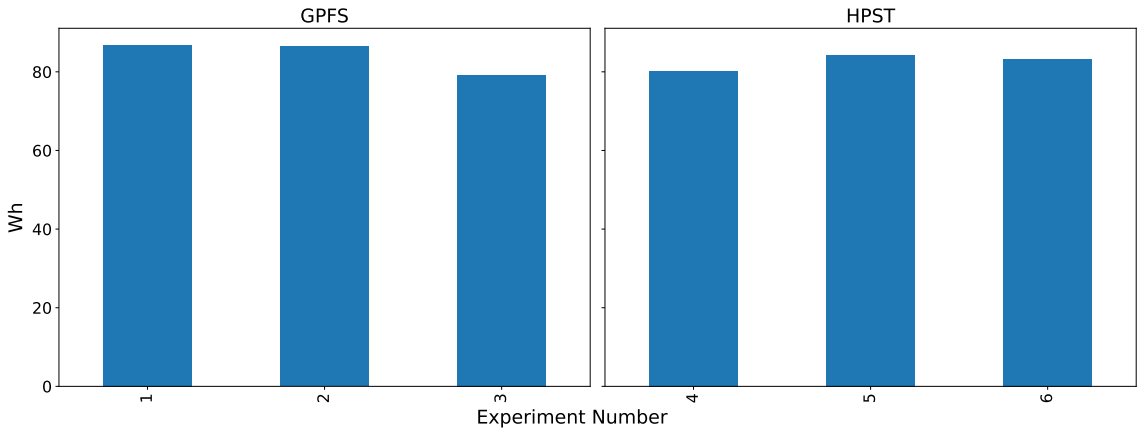



Figure 68: AP5 JUWELS Booster Energy: Total GPU energy consumption during the training phase *ap5-jwb-energy*

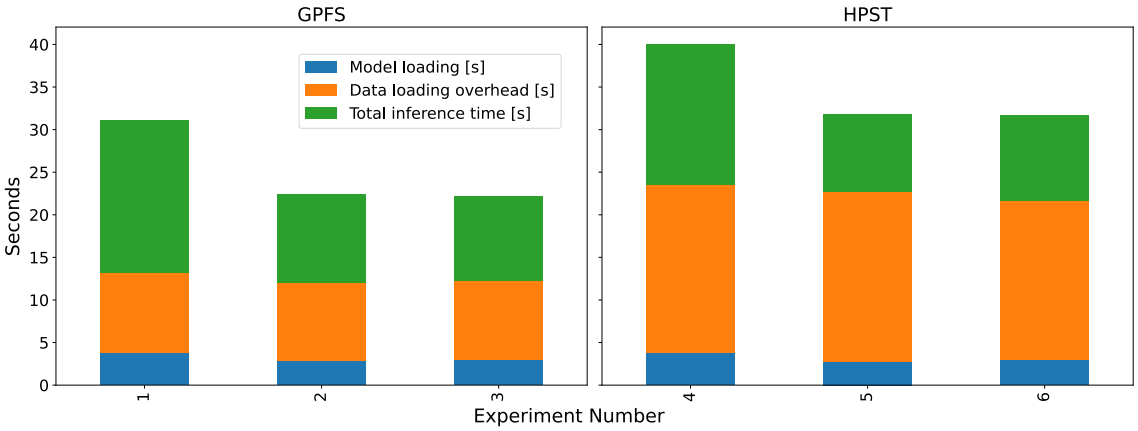
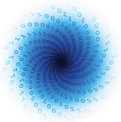



Figure 69: AP5 JUWELS Booster Inference Runtime: Runtime and relative share for multiple experiments *ap5-jwb-inf-runtime-share*



### 4.5.3 JUWELS Cluster

In the AP5 on the Juwels Cluster, a total of 6 runs were conducted to test the training phase, with 3 runs using the SCRATCH disk and the other 3 using the CSCRATCH disk. Additionally, other six runs were performed with the same configuration to test the inference phase.

#### 4.5.3.1 Training

In Figure 70, we present the time distribution for each experiment between data loading and training. In the case of the GPFS filesystem, the average total runtime is $5358.5 \pm 13.2$, with the training process taking on average $5293.6 \pm 10.9$ s, and the data loading requiring $3590.4 \pm 32.1$ s. On the other hand, for the CSCRATCH filesystem, the total runtime is $5306.2 \pm 48.6$, with a mean training time and data loading time of $5245.3 \pm 55.8$ s and $3676.9 \pm 48.7$ s, respectively.
Overall, we observe that CSCRATCH is slightly faster during the training phase, while it is slightly slower during the data loading phase. However, the execution time between the two filesystems is homogeneous in general. It is important to note that compared to the Booster case, the Juwels Cluster takes almost twice as long to execute the application.

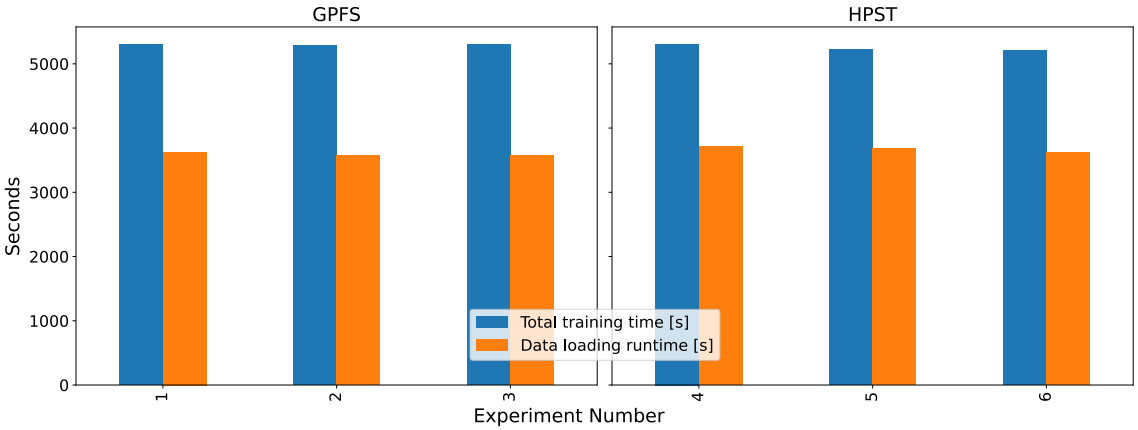



Figure 70: AP5 JUWELS Cluster Runtime: Runtime and relative share for multiple experiments during the training phase. *ap5-jwc-runtime-share*

In Figure 71, we can see the time taken to train the first epoch and the average training time per epoch. Across all experiments, the first epoch takes slightly longer than the average training time per epoch. In the GPFS case, the average first epoch training time is $1114.7 \pm 5.5$ s, while the average training time per epoch is $1058.7 \pm 13.2$ s. In the HPST case, these values are $1100.0 \pm 10.8$ s and

1049.1 ± 11.2 s, respectively. The difference between the first epoch training time and the average training time per epoch is almost the same for both GPFS and HPST, around 53.5 ± 5.1 s. Overall, the average training time per epoch and the first epoch training time are slightly lower in the HPST case.

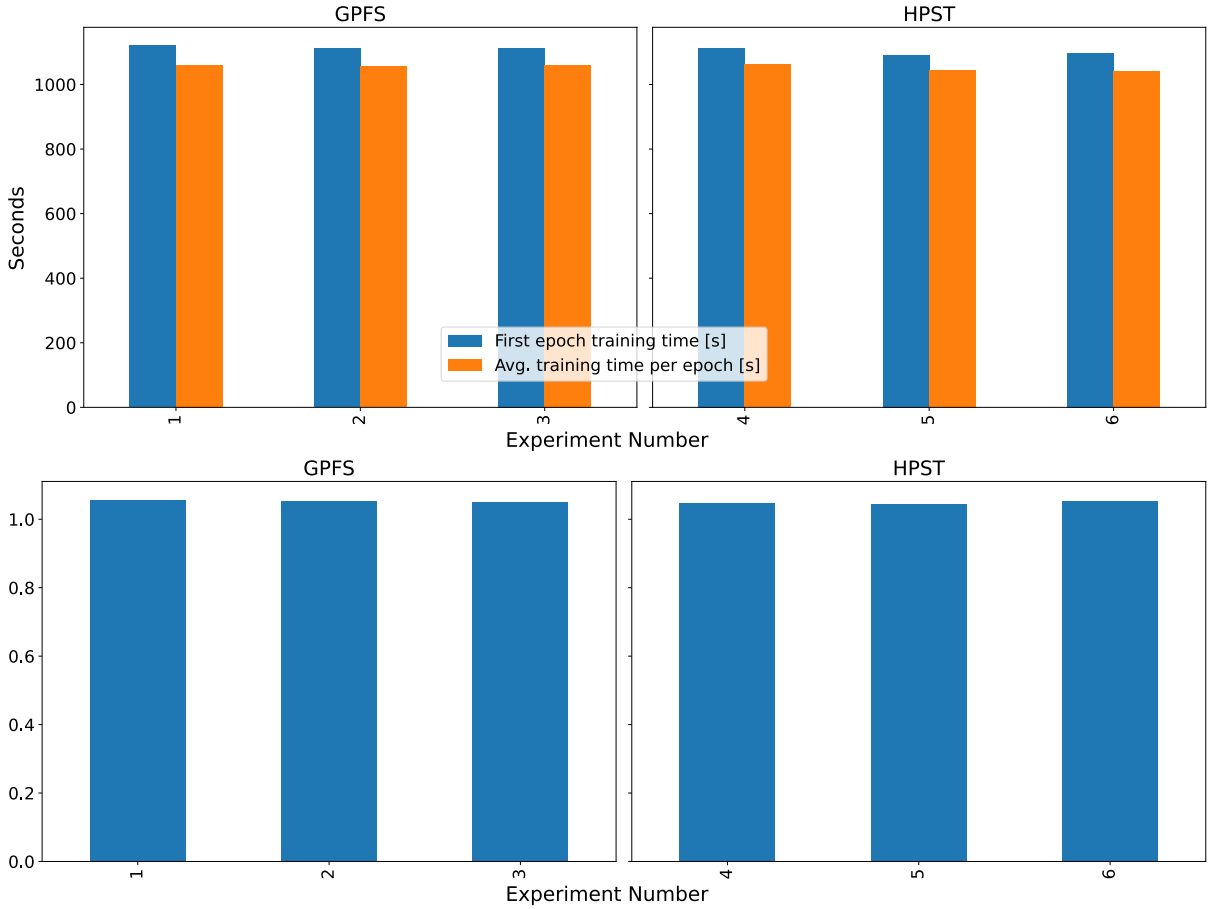



Figure 71: AP5 JUWELS Cluster Epoch Time: Comparison of time for first epoch and average time for an epoch (top); ratio of both quantities (bottom). *ap5-jwc-epoch-time*

The power consumption of the GPUs is presented in Figure 72, showing a consistent and location-independent trend. Comparing with the results in Figure 68, it can be observed that the newer A100 GPUs in the Booster outperform the older V100s and consume less energy: specifically, in the Juwels Cluster, the average GPU energy consumption is 131.05 ± 2.78 Wh, which is 1.5 times greater than the energy consumption of the Booster case.

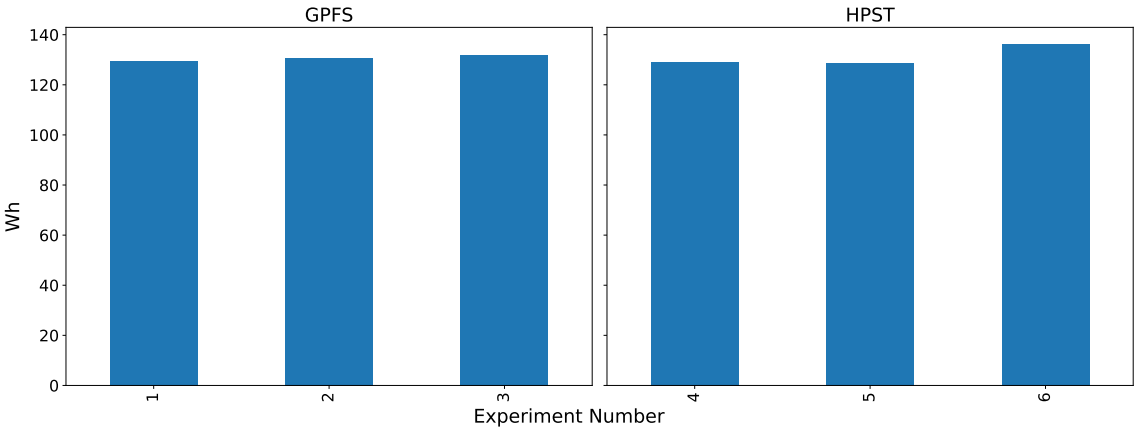



Figure 72: AP5 JUWELS Cluster Energy: Total GPU energy consumption during the training phase. *ap5-jwc-energy*

#### 4.5.3.2 Inference

In Figure 73, we can see that the total inference time per run ranges from 15.58 to 16.02 seconds (in average 15.9 ± 0.3 s), with data loading time being the most significant contributor, varying from 27.43 to 43.12 seconds (in average 34.6 ± 7.9 s), and the model loading time being relatively low, ranging from 6.79 to 7.41 seconds (in average 7.1 ± 0.3 s) . Loading the data consumes over 50% of the total inference time in each run. The decreasing trend in the data loading phase observed in the SCRATCH case needs further investigation.
In the CSCRATCH case, the model loading time ranges from 7.05 to 7.62 seconds, with an average of 7.3 ± 0.3 seconds. The data loading time varies from 32.60 to 34.22 seconds, with an average of 33.2 ± 0.9 seconds. Finally, the total inference time ranges from 15.59 to 17.08 seconds, with an average of 16.1 ± 0.8 seconds.
The results suggest that the model loading time and total inference execution time are similar for both SCRATCH and CSCRATCH. However, data loading time remains the bottleneck in the inference process for both cases, with the CSCRATCH case showing more uniformity. Thus, it can be concluded that data loading is the most time-consuming phase of the inference process, whereas inference and model loading are relatively fast.

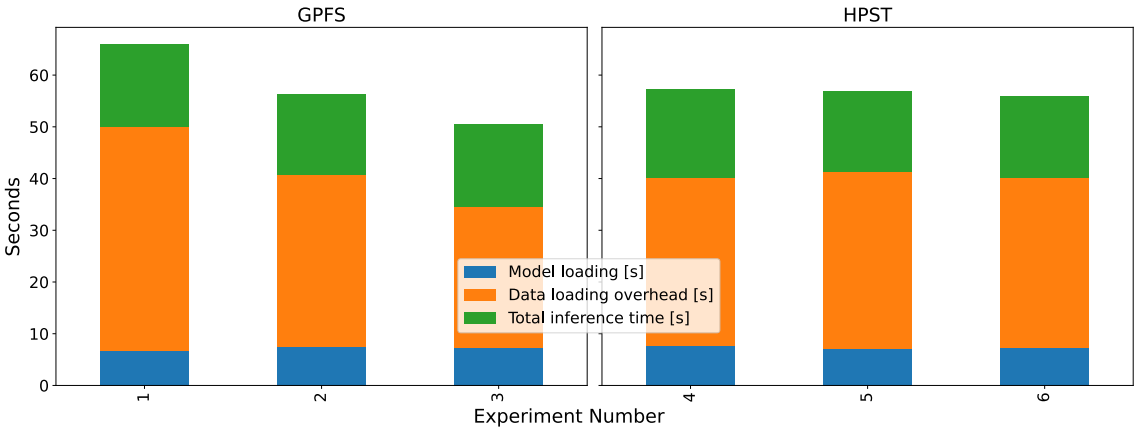



Figure 73: AP5 JUWELS Cluster Inference Runtime: Runtime and relative share for multiple experiments durnig the inference phase. *ap5-jwc-inf-runtime-share*

### 4.5.4 E4 Intel System

In the AP5, we conducted three runs using the NFS filesystem on a single node with one GPU in the Intel system. These runs aimed to test the training and inference performance.

#### 4.5.4.1 Training

As reported in Figure 74, the average total runtime of the three runs is $3176.9 \pm 210.5$, which is characterized by the training and data loading processes that occur in parallel. The training process takes up $3148.9 \pm 201.9$ on average and data loading requiring $2751.5 \pm 248.1$
Comparing these results with the SCRATCH Booster case, we can see that the loading data times in the Booster case are significantly shorter (almost 29% shorter) than in the Intel case. The total training times are generally shorter, although the difference is not as pronounced: in average, the difference between the total training time on Intel system and on Booster system amounts to 266.8 s.
Referring to the information presented in Figure 75, it is noticeable that the first epoch training time and the average training time per epoch of each experiment are quite similar, as indicated by the "Ratio" plot, which shows values close to 1. Specifically, the average training time per epoch is $629.8 \pm 40.4$ s, while the average first epoch training time is $651.8 \pm 43.9$ s. The difference between the two values is about $21.9 \pm 4.8$ s on average.
The plot of energy consumption reported in Figure 76 follows a similar trend to

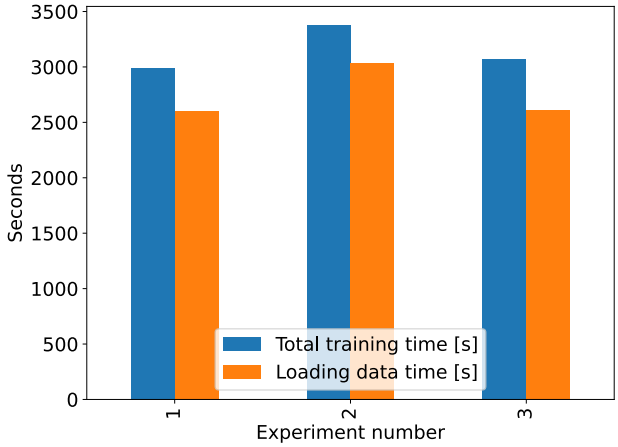



Figure 74: AP5 E4 Intel System Runtime: Runtime and relative share for multiple experiments during the training phase *ap5-e4i-runtime-share*

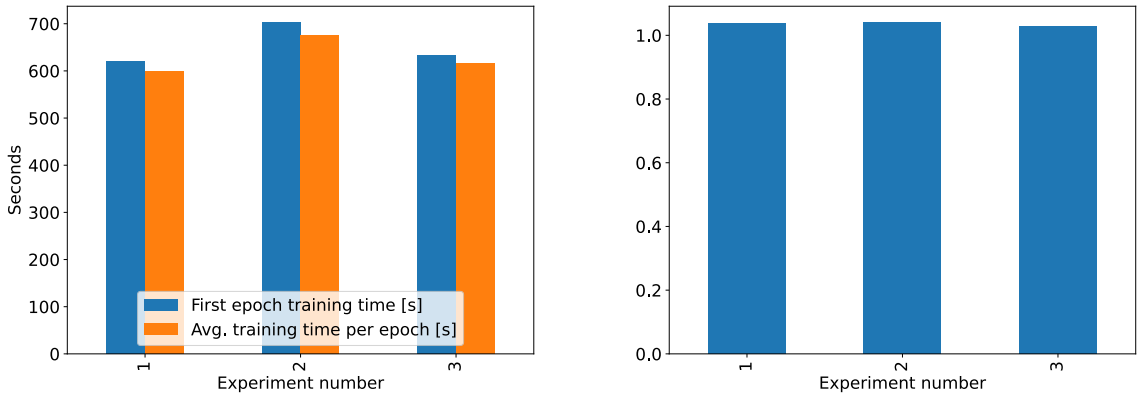



Figure 75: AP5 E4 Intel System Epoch Time: Comparison of time for first epoch and average time for an epoch (left); ratio of both quantities (right) *ap5-e4i-epoch-time*

that of runtime, with the second test showing the highest node consumption of 922.9 Wh. The three experiments reported energy consumption values of 722.5 Wh, 922.9 Wh, and 743.5 Wh, respectively. On average, the power consumption of a node in the Intel system is $796.3 \pm 110.1$ Wh.

As reported in Figure 77, we can see that there is significant variation between the values. Experiment 2 has the highest action value of 11348.04 MJs, while Experiment 1 has the lowest action value of 7848.17 MJs. Experiment 3 has an intermediate action value of 8292.11 MJs. The difference between the highest and lowest

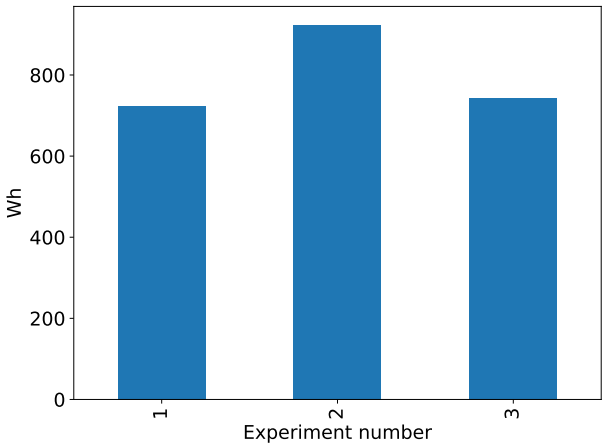



Figure 76: AP5 E4 Intel System Energy: Total node energy consumption *ap5-e4i-energy*

Action values is substantial, with a range of 3499.87 MJs, with an average action value that is $9156.44 \pm 1500.39$ MJs.

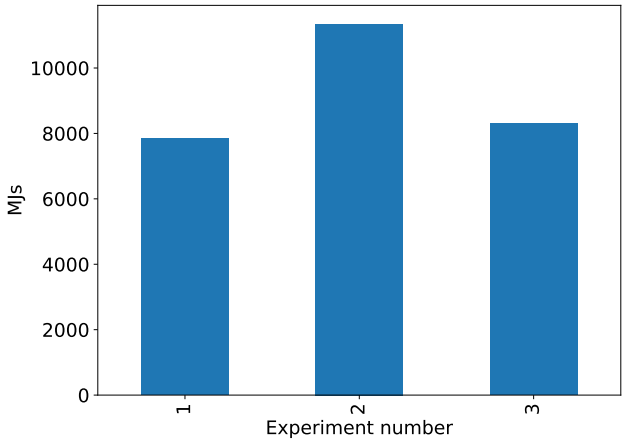
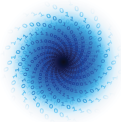



Figure 77: AP5 E4 Intel System Action: Action values (in MJs) for three applications during the training phase. *ap5-e4i-action*

#### 4.5.4.2 Inference

According to Figure 78, there is a decreasing trend in the total runtime, with the model loading being the only component that shows a constant time between the three runs. In particular, data loading is the predominant component, accounting



for 70.6%, 74.5%, and 76.5% of the total runtime in the three runs, respectively. The remaining runtime is spent on total inference time, which covers 27.5% of the runtime in experiment 1 and 19.9% of the total runtime in runs 2 and 3.
While runs 2 and 3 show homogeneous values between the three components, run 1 exhibits much higher values for data loading time and total inference time. Specifically, runs 2 and 3 have an average data loading time and total inference time of $59.8 \pm 1.0$ s and $15.8 \pm 0.6$ s, respectively, while run 1 has a data loading time of 110.8 s and a total inference time of 43.2 s.

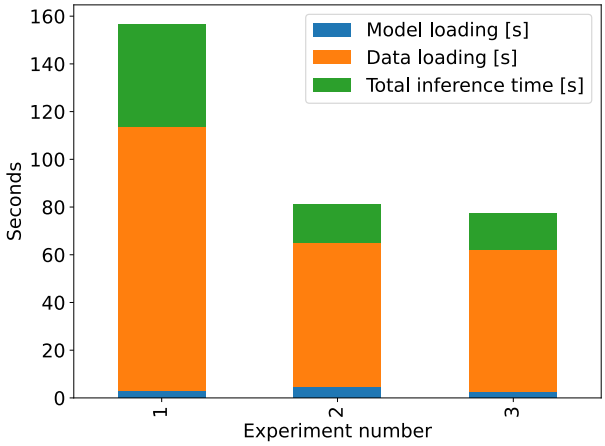
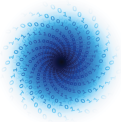



Figure 78: AP5 E4 Intel System Inference Runtime: Runtime and relative share for multiple experiments for the inference phase *ap5-e4i-inf-runtime-share*



### 4.5.5 E4 AMD System

On the AMD system, within the AP5, three experiments using the NFS filesystem, with each run utilizing one node and one GPU were conducted. Additionally, three runs were performed to test the inference process.

#### 4.5.5.1 Training

Looking at Figure 79, we can see that three experiments have been conducted to test the AMD system. For all the experiments, the total runtime is mainly split between the data loading and the training process. The data loading and training phases appear to have the same proportion of runtime across all three experiments, indicating consistency in the system's performance. On average, the data loading phase takes $1440.8 \pm 14.6$ s, while the total training time takes $4962.4 \pm 24.6$ s. Both the loading data time and the total training time are relatively consistent across all experiments. However, compared to the other machines, data loading on the AMD system is the least time-consuming. On the other hand, the training phase is much slower.

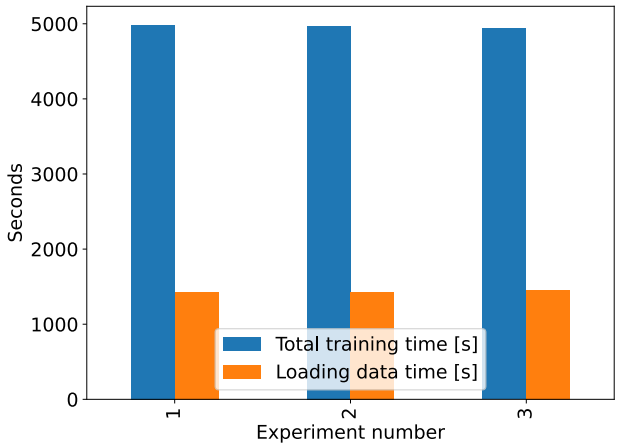



Figure 79: AP5 E4 AMD System Runtime: Runtime and relative share for multiple experiments for the training phase *ap5-e4a-runtime-share*

In Figure 80, it can be observed that there is not a significant difference between the first epoch training time and the average epoch training time for each experiment. The average epoch training time for the three experiments is $992.5 \pm 4.9$ s, which is only slightly shorter than the first epoch training time of $1026.3 \pm 5.5$ Comparing these results with the Intel system, it can be seen that the AMD system

takes 1.57 times longer in epoch training, both in terms of average training time per epoch and first epoch training time.

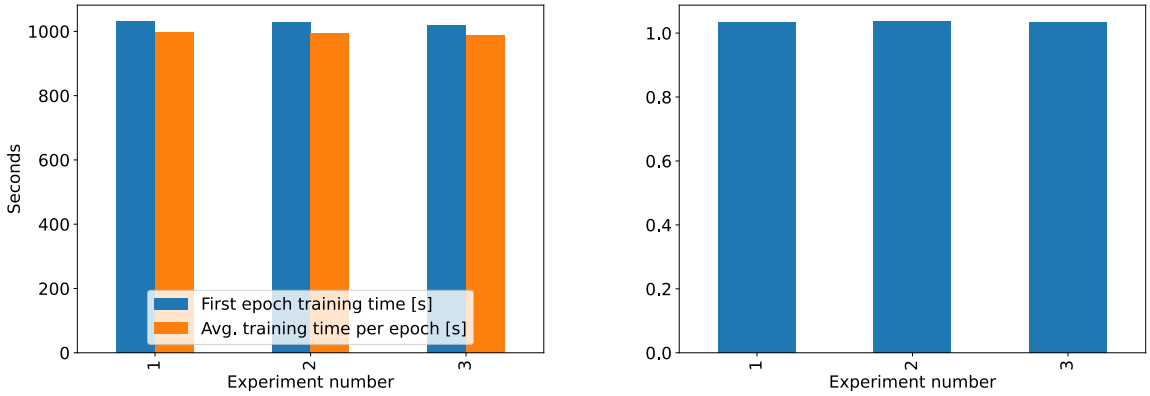



Figure 80: AP5 E4 AMD System Epoch Time: Comparison of time for first epoch and average time for an epoch (left); ratio of both quantities (right) *ap5-e4a-epoch-time*

Energy consumption, as shown in Figure 81, is considerably higher in the AMD case compared to the Intel case. The average energy consumption of the AMD system across the three runs is 1021.80 ± 4.93 Wh, which is approximately 1.28 times higher than the average energy consumption of the Intel system (796.3 ± 110.1 Wh). Also, it can be seen that the energy consumption remained consistent across the three runs.

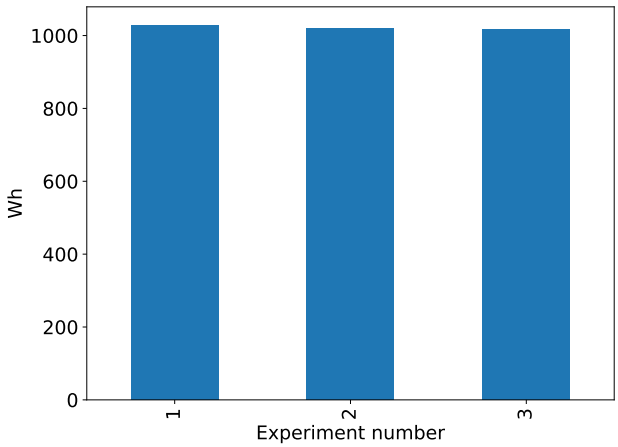
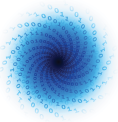



Figure 81: AP5 E4 AMD System Energy: Total node energy consumption during training phase *ap5-e4a-energy*



Looking at the action values reported in Figure 82, we can see that they are all relatively close to each other, with Experiment 1 having the highest action value of 18507.79 MJs and Experiment 3 having the lowest action value of 18294.23 MJs. Experiment 2 has an action value of 18332.00 MJs, which is very close to the values of the other two experiments. The average action for all three experiments is $18\,378.00 \pm 113.97$ MJs, indicating that the data is relatively consistent, and any differences between the experiments may not be statistically significant.

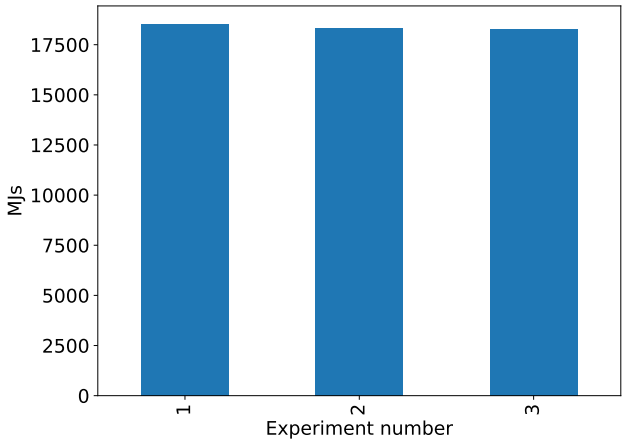



Figure 82: AP5 E4 AMD System Action: Action variation during training phase *ap5-e4a-action*

#### 4.5.5.2 Inference

The three experiments aimed at testing inference exhibit remarkable homogeneity in terms of the distribution of runtime components, with model loading, data loading, and total inference time showing consistent proportions across all experiments.

Model loading is the smallest component in terms of runtime, averaging at $2.80 \pm 1.02$ s. Total inference time, which covers about 21.3% of the runtime, amounts to an average of $14.8 \pm 0.1$ s. The majority of the runtime is dedicated to data loading, which represents a bottleneck for inference with a runtime of approximately $51.6 \pm 0.1$ s, accounting for 74.6% of the total runtime.

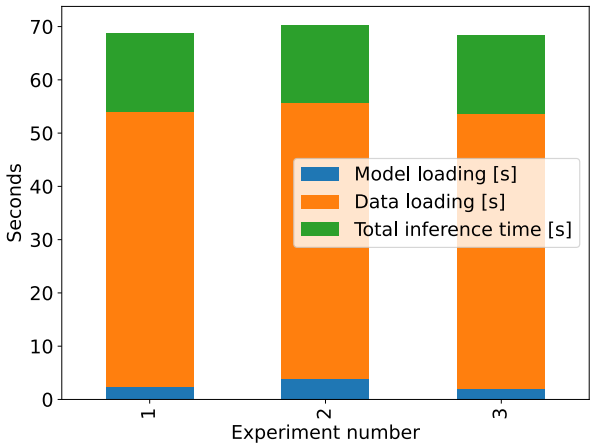


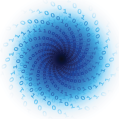

Figure 83: AP5 E4 AMD System Inference Runtime: Runtime and relative share for multiple experiments during inference phase *ap5-e4a-inf-runtime-share*

### 4.5.6 Results

In the context of AP5, we conducted performance tests on both SCRATCH and CSCRATCH filesystems in JSC systems, while only NFS filesystem was tested in E4 machines. Similar to AP3, the AMD system shows better performance in data loading, while Juwels Booster is the best system for training and inference runtime. When using CSCRATCH instead of SCRATCH, Juwels Booster shows a 20% increase in data loading time during training phase, but training time remains relatively constant. On the other hand, in the Juwels Cluster case, both training and loading times are consistent.

Comparing JSC systems, we find that V100 GPUs in Juwels Cluster consume 1.6 times more energy than A100s in Juwels Booster, and this is due to the longer application execution time in the Cluster case (about 1.8 times longer than in the Booster case).

Despite the AMD system being more efficient in terms of data loading, the Intel system exhibits higher computational efficiency and energy savings. This can be seen by comparing the Action values, which average at 9162.8 and 18378.0 MJs for Intel and AMD cases, respectively.



## 4.6 AP 6

### 4.6.1 Notes

| Training dataset | Memory validation dataset | Training samples | Input shape sample | batch size |
|---|---|---|---|---|
| 1.7 GB<br>1.4 GB | - | 35 064 | (128, 128, 3) | 750 |

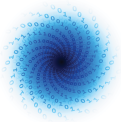
| Trainable parameters | Non-trainable parameters | Loss function | Experimental notes |
|---|---|---|---|
| - | - | cross-entropy | 100 epochs,1.7 GB dataset on JSC<br>50 epochs, 1.4 GB dataset on E4 |

| Data formats | Frameworks (to be) used |
|---|---|
| NetCDF, TIFF | PyTorch |

Previously, AP6 did not implement neural networks (NNs), but used rather conservative ML algorithms (Principal Component Analysis and K-means or HDBSCAN clustering) to try achieve a linear dimensionality reduction of the data. In a new approach, AP6 aims to achieve a non-linear dimensionality reduction using a siamese NN structure called DeepCluster v2 (DCv2) that allows self-supervised clustering of visual features [1, 2]. The algorithm is implemented in the open-source VISSL library from Facebook Artificial Intelligence Research (FAIR) [3].
DCv2 makes use of two branches to achieve self-supervised clustering:

1. The first branch runs the input image through a ResNet-50 [4] followed by a simple multi-layer perceptron (MLP). The output feature vector of the MLP is then used to apply a spherical K-means on the feature vector: the algorithm is initiated with a random set of centroids, and each sample gets a label assigned based on the lowest cosine similarity between its feature vector and the cluster centers.

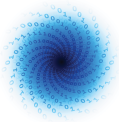
2. The second branch is almost identical to the first branch: it runs the image through a ResNet-50 and a MLP. Here, the resulting feature vector is then used – together with the output of the upper branch – to calculate the cross-entropy loss function. The network then tries to minimize this loss function by



sequentially adjusting the parameters in each layer through back propagation. The resulting set of parameters is then used for the next iteration.

The algorithm requires input images of size $128 \times 128$. However, each branch uses an individual random crop that is 75% of the original image size, hence $96 \times 96$. Here, we use hourly data from the IFS HRES model provided in the form of netCDF files. Since the algorithm requires image data as input though, the training dataset consists of TIFF images created from the netCDF source files. In order to convert the data to TIFF, we extract three physical quantities (absolute wind speed, relative humidity, and geopotential height at 500 hPα) on the $0.1°$ grid ranging 0–$11.9$°E and $43.77$–$55.76$°N, resulting in a $128 \times 128$ pseudo-RGB image, where each quantity represents one of the color channels. For the conversion to RGB channels using floating-point values in the range $[0, 1]$, each value in the physical fields is normalized by scaling using the global spatio-temporal minimum and maximum of the respective field.
AP6 makes use of Apptainer containers to run the application. For technical reasons it was not possible to build the image for ROCm or ARM-based systems. Thus, these systems could not be benchmarked with AP6.
In the following paragraphs, we are going to analyze the training performance of the AP6 benchmark.
Data for the benchmarks which are shown here can be found in appendix 6.6.
The application code is accessible on GitHub[4].

[4] https://github.com/4castRenewables/maelstrom-a6/tree/main/mlflow/deepclusterv2



| Experiment number | Number of Nodes | Number of GPUs |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 1 | 4 |
| 4 | 2 | 4 |
| 5 | 4 | 4 |

Table 1: Configuration of each experiment number performed on Juwels Booster

### 4.6.2 JUWELS Booster

In AP6, we used Juwels Booster to conduct five training runs with different configurations. The configuration of each experiment is summarized in Table 1, which includes the number of nodes and GPUs used in each case. Experiment 1 involved a single node and GPU, while Experiment 5 utilized four nodes, each with four GPUs. In Figure 84, it can be observed that the total runtime varies depending on the number of nodes and GPUs used in the experiment. Experiment 1, which employed only 1 node and 1 GPU, took 9496.6 seconds to complete. Experiment 2, which used 1 node and 2 GPUs, had a longer total runtime than experiment 1 but was shorter than experiment 3, which utilized 1 node and 4 GPUs. Experiment 3 has the longest total runtime at 15031.9 seconds.
On the other hand, experiment 4, which utilized 2 nodes and 4 GPUs, had a shorter total runtime than experiment 3. Finally, experiment 5, which employed 4 nodes and 4 GPUs, had the shortest total runtime at 5278.4 seconds. This suggests that increasing the number of nodes and GPUs can lead to significant improvements in performance. However, it should be noted that experiment 2 and 3 also experienced a runtime increase despite using more resources than experiment 1. Therefore, the actual performance improvement is achieved when more than one node is used, while the worst performance is reported by the case using 4 GPUs on a single node.
As shown in Figure 85, by comparing the first epoch training time and the average epoch training time in AP6's experiments, we can see that the ratio between these two quantities changes as the number of nodes and GPUs used changes. Specifically, in experiments 1 and 5, where only one GPU per node is used, the first epoch training time is greater than the average epoch training time. However, in the other experiments where more than one GPU per node is used, the average epoch training time is greater than the first epoch training time. This difference could be due to a contention delay between the resources.
The plot in Figure 86 shows the average energy consumption of a single GPU for

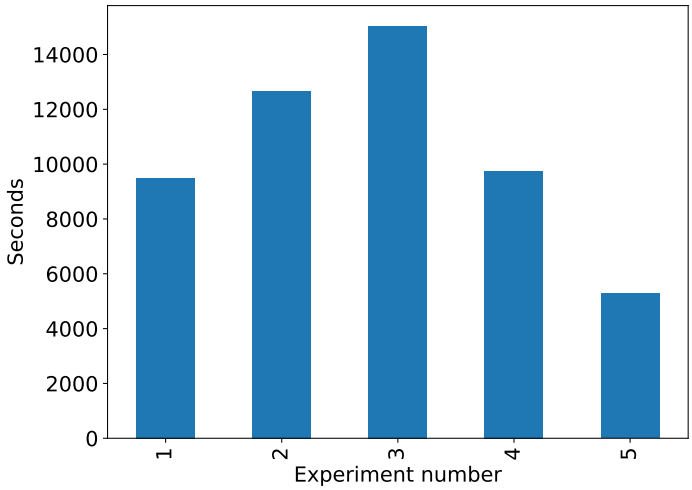



Figure 84: AP6 JUWELS Booster Runtime: Runtime and relative share for multiple experiments *ap6-jwb-runtime-share*

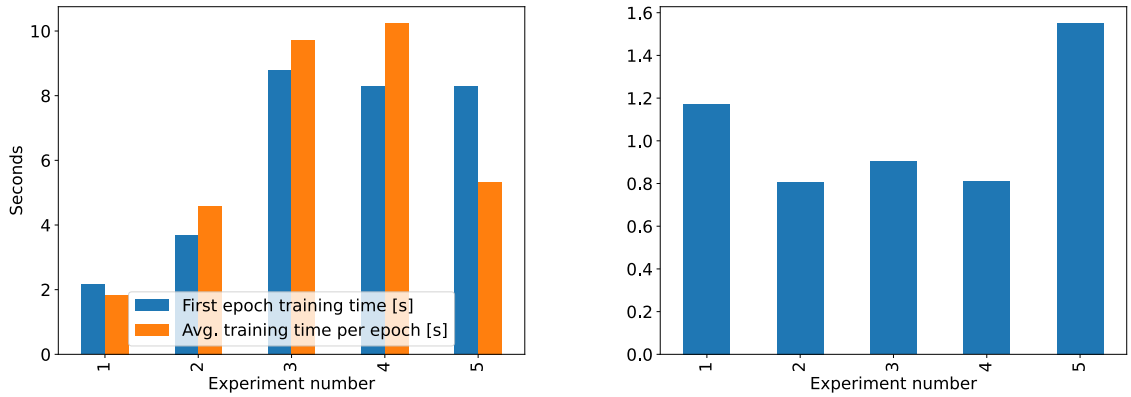



Figure 85: AP6 JUWELS Booster Epoch Time: Comparison of time for first epoch and average time for an epoch (left); ratio of both quantities (right) *ap6-jwb-epoch-time*

each of the five experiments. The plot indicates that there is a significant variation in GPU energy consumption across the different experiments. Experiment 3 had the highest mean GPU energy consumption, with an average of 361.5 Wh, while experiment 5 had the lowest mean GPU energy consumption, with an average of 119.97 Wh. The remaining experiments (1, 2, and 4) had mean GPU energy consumption values of 180.86 Wh, 255.71 Wh, and 219.98 Wh, respectively.

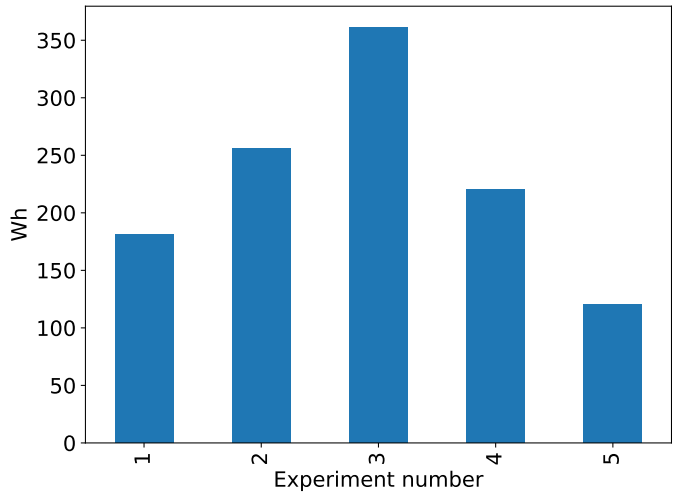



Figure 86: AP6 JUWELS Booster Energy: Total GPUs energy consumption (left); peak and average GPUs power draw (right) *ap6-jwb-energy*

| Experiment number | Number of Nodes | Number of GPUs |
|---|---|---|
| 1 | 1 | 1 |
| 2 | 1 | 2 |
| 3 | 1 | 4 |
| 4 | 2 | 4 |
| 5 | 4 | 4 |

Table 2: Configuration of each experiment number performed on Juwels Cluster

### 4.6.3 JUWELS Cluster

In AP6, Juwels Cluster was used to test five runs with different configurations during the training phase. The table 2 shows the configuration of each experiment, specifying the number of nodes and GPUs used in each case. The experiments range from a single node and GPU (Experiment 1) to four nodes with four GPUs each (Experiment 5).

The total runtime of five different experiments with varying configurations in terms of the number of nodes and GPUs used for training is reported in Figure 87.

Experiments 1, 2, and 3 were performed on a single node and used 1, 2, and 4 GPUs, respectively. The total runtime for these experiments increased as the number of GPUs used increased, with Experiment 3 having the longest runtime of 18,581.3 seconds.

Experiments 4 and 5 were performed on 2 and 4 nodes, respectively, and used 4

GPUs each. The total runtime for these two experiments decreased as the number of nodes used increased, with Experiment 5 having the shortest runtime of 5,763.2 seconds. These results suggest that using more nodes can lead to a shorter total runtime.

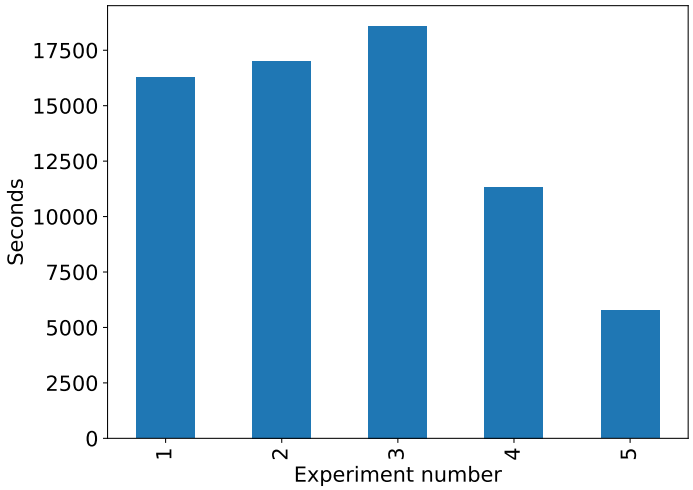



Figure 87: AP6 JUWELS Cluster Runtime: Runtime for multiple experiments *ap6-jwc-runtime*

Figure 88 shows the epoch training times for five experiments. In the single node configuration, as the number of GPUs increases, the training times generally increase, as seen in experiments 1-3. Additionally, the first epoch training time is typically shorter than the average training time per epoch.
In the multiple nodes case, experiment 4 had a first epoch training time of 6.758 seconds and an average training time per epoch of 13.370 seconds, while experiment 5 had a lower average training time per epoch of 6.711 seconds and a first epoch training time of 8.592 seconds. In general, as the number of GPUs and nodes increases, the average training time per epoch also increases, except for experiment 5. Experiment 4, with two nodes and four GPUs, had the highest average training time per epoch of 13.370 seconds, while Experiment 5, with four nodes and four GPUs, had an average training time per epoch of 6.711 seconds.
The energy consumption of the GPU in the five experiments is shown in Figure Figure 89. The energy consumption ranges from 110.717 Wh (Experiment 5) to 395.369 Wh (Experiment 3). The results suggest that in the single node configuration, increasing the number of GPUs on a node also increases the average energy consumption of a single GPU. However, using more nodes and GPUs can result in a

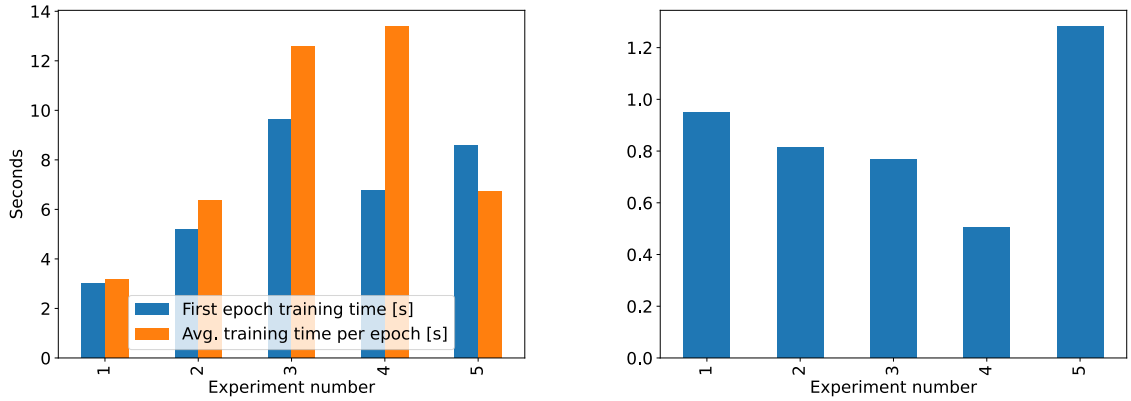



Figure 88: AP6 JUWELS Cluster Epoch Time: Comparison of time for first epoch and average time for an epoch (left); ratio of both quantities (right) *ap6-jwc-epoch-time*

lower average energy consumption per GPU, as observed in Experiment 5.

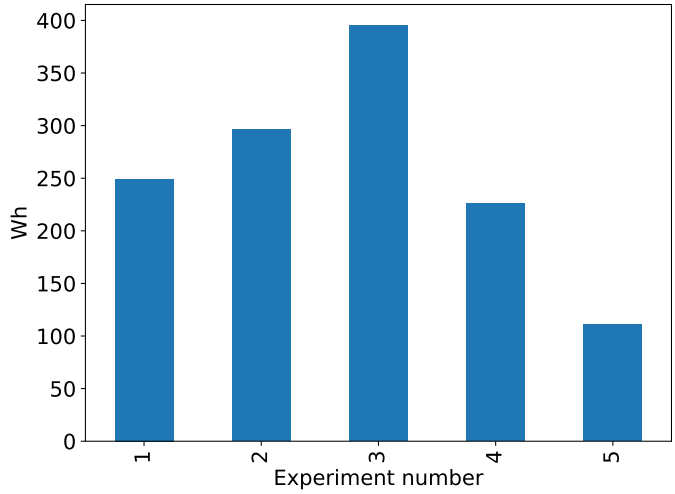
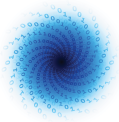



Figure 89: AP6 JUWELS Cluster Energy: Total node energy consumption (left); peak and average node power draw (right) *ap6-jwc-energy*

### 4.6.4 E4 Intel System

Within AP6, two experiment were performed on the E4 Intel System. The configuration and the the available metrics of this analysis are reported in 3.
The table provides insights into the impact of different configurations on training



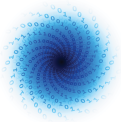
| Experiment number | #Nodes | #GPUs | Total runtime [s] | Avg. training time per epoch [s] | Node energy consumption [Wh] |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 6240.74 | 2.46 | 1041.18 |
| 2 | 2 | 1 | 3936.07 | 2.85 | 655.02 |

Table 3: Training on E4 Intel System

performance and energy consumption. The use of two nodes with one GPU each resulted in a significantly shorter total runtime of 3936.077 seconds, compared to the single node and single GPU configuration, which took 6240.740 seconds, representing a 37% decrease. This demonstrates that increasing the number of nodes and GPUs can improve training performance and reduce overall runtime.
Regarding the average training time per epoch, there is a relatively small difference between the two experiments. However, Experiment 2 takes slightly longer than Experiment 1 and this aspect needs further checks and investigation.
Looking at the Node energy consumption values, the reported value for Experiment 2 corresponds to the average energy consumption of both nodes. This experiment was more energy-intensive due to the use of two nodes. However, it was able to complete the same task in less time due to the increased resources available, resulting in a more efficient use of energy.

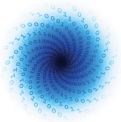
### 4.6.5 Results

In AP6, the runtime was measured on JUWELS Booster, JUWELS Cluster, and the Intel system using different hardware configurations. In the JSC context, a total of five tests were conducted for each machine, ranging from using a single node with a single GPU to four GPUs distributed on four different nodes. Experiment 5, which employs four nodes with one GPU each, was found to have the best performance in both cases. When comparing the runtime and energy consumption of experiment 5 on both JSC systems, we found that the Booster case had a slightly faster runtime, while the energy consumption was similar.
In the E4’s Intel System, we conducted two tests on a single machine, one using a single node with one GPU and the other using two nodes with one GPU each. Based on our observations, the latter configuration performed better in terms of both runtime and energy consumption.



# 5 Conclusion

The work done in this deliverable showed strong cooperation between partners providing the computing systems and those developing the applications, often running applications in a coordinated manner between developers and hardware engineers to verify all the physical parameters of the computing systems.
For each application, data were collected and presented in the form of graphs in the various dedicated sections.
When it has been possible to parallelize application training across multiple GPUs, as in the case of AP2 on Julich's machines, a significant increase in performance has been seen; we can say that this is one of the path to follow in future developments. As the previous deliverable has shown a strong impact of IO on training and inference time, great care was taken to test the performance of different filesystems (NFS, local NVMe, GPFS, HPST), and it was found that the optimal choice of the file system can reduce data loading time during training and reduce the total time-to-solution by up to 54% (see AP1). It was possible to use the newly suggested "Action" score metric for the AP1, AP2, AP3 and AP5. Results show that Action is minimized for the Intel system which indicates an advantage when using Intel processors. Further studies will be performed to understand the reasons for this in more detail.



# References

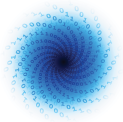
[1] Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Matthijs Douze. Deep clustering for unsupervised learning of visual features. *CoRR*, abs/1807.05520, 2018.

[2] Mathilde Caron, Ishan Misra, Julien Mairal, Priya Goyal, Piotr Bojanowski, and Armand Joulin. Unsupervised learning of visual features by contrasting cluster assignments, 2021.

[3] Priya Goyal, Quentin Duval, Jeremy Reizenstein, Matthew Leavitt, Min Xu, Benjamin Lefaudeux, Mannat Singh, Vinicius Reis, Mathilde Caron, Piotr Bojanowski, Armand Joulin, and Ishan Misra. Vissl. `https://github.com/facebookresearch/vissl`, 2021.

[4] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

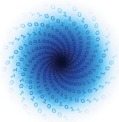
# 6 Appendix



| Notes | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | CSCRATCH | CSCRATCH | CSCRATCH |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Experiment number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
| Job ID | 6734785 | 6737281 | 6737357 | 6734793 | 6737282 | 6737358 | 6736672 | 6737283 | 6737359 | 6754538 | 6754539 | 6754540 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 2 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 4 |
| #MPI tasks | 1 | 1 | 1 | 2 | 2 | 2 | 4 | 4 | 4 | 4 | 4 | 4 |
| #CPUs per task | 48 | 48 | 48 | 24 | 24 | 24 | 12 | 12 | 12 | 12 | 12 | 12 |
| Total runtime | 694.34 | 681.94 | 673.29 | 536.18 | 511.58 | 491.48 | 460.42 | 462.29 | 482.59 | 487.79 | 503.94 | 489.11 |
| Total training time | 694.03 | 681.65 | 673.00 | 535.90 | 511.20 | 491.15 | 460.10 | 461.87 | 482.15 | 487.43 | 503.59 | 488.65 |
| Avg. training time per epoch | 231.20 | 227.13 | 224.23 | 178.53 | 170.33 | 163.60 | 153.29 | 153.82 | 160.65 | 162.30 | 167.70 | 162.71 |
| Performance [GB/s] | 1.43 | 1.45 | 1.47 | 1.85 | 1.94 | 2.01 | 2.15 | 2.14 | 2.05 | 2.03 | 1.96 | 2.02 |
| First epoch training time | 284.49 | 274.57 | 271.35 | 233.07 | 212.49 | 212.07 | 216.50 | 218.20 | 216.58 | 239.38 | 249.90 | 241.73 |
| Min. training time per epoch | 194.27 | 193.73 | 193.75 | 142.06 | 142.24 | 125.34 | 95.92 | 91.96 | 94.56 | 102.53 | 102.57 | 99.11 |
| Max. training time per epoch | 284.49 | 274.57 | 271.35 | 233.07 | 212.49 | 212.07 | 216.50 | 218.20 | 216.58 | 239.38 | 249.90 | 241.73 |
| Avg. training time per batch | 0.98 | 0.96 | 0.95 | 0.76 | 0.72 | 0.69 | 0.65 | 0.65 | 0.68 | 0.69 | 0.71 | 0.69 |
| Final training loss | 0.24 | 0.25 | 0.25 | 0.23 | 0.26 | 0.24 | 0.38 | 0.25 | 0.25 | 0.28 | 0.25 | 0.26 |
| Final validation loss | 0.27 | 0.27 | 0.29 | 0.28 | 0.29 | 0.32 | 0.33 | 0.32 | 0.29 | 0.29 | 0.32 | 0.29 |
| Max CPU memory per MPI task [GB] | 129.48 | 130.26 | 125.72 | 118.35 | 118.16 | 124.37 | 114.98 | 119.46 | 116.99 | 108.25 | 110.97 | 115.12 |
| MAX GPU memory per MPI task[GB] | 27.07 | 27.21 | 27.07 | 27.30 | 27.30 | 27.19 | 27.40 | 27.40 | 27.40 | 27.19 | 27.07 | 27.19 |
| Node ID | jwb0727 | jwb0215 | jwb0093 | jwb0388 | jwb0216 | jwb0119 | jwb0150 | jwb0217 | jwb0138 | jwb1241 | jwb1083 | jwb1084 |
| Max. GPU power [W] | 327.61 | 341.05 | 313.16 | 338.44 | 354.09 | 318.44 | 323.11 | 330.71 | 351.86 | 346.47 | 335.99 | 325.39 |
| Avg. GPU power [W] | 78.37 | 81.42 | 85.73 | 92.84 | 102.03 | 118.16 | 110.40 | 121.57 | 135.17 | 135.30 | 109.09 | 98.07 |
| GPU energy consumption [Wh] | 15.12 | 15.42 | 16.03 | 13.83 | 14.50 | 16.13 | 14.12 | 15.61 | 18.12 | 18.33 | 15.27 | 13.32 |

Table 4: AP1 JUWELS Booster training benchmark

## 6.1 AP 1

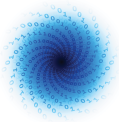


| Notes | SCRATCH | SCRATCH | SCRATCH |
|---|---|---|---|
| Experiment number | 1 | 2 | 3 |
| Job ID | 6737400 | 6737464 | 6737465 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 |
| #CPUs per task | 40 | 40 | 40 |
| Total runtime | 1531.54 | 1518.48 | 1520.98 |
| Total training time | 1531.19 | 1518.21 | 1520.65 |
| Avg. training time per epoch | 510.23 | 505.95 | 506.79 |
| Performance [GB/s] | 0.65 | 0.65 | 0.65 |
| First epoch training time | 526.40 | 525.98 | 525.67 |
| Min. training time per epoch | 487.51 | 482.33 | 487.83 |
| Max. training time per epoch | 526.40 | 525.98 | 525.67 |
| Avg. training time per batch | 1.08 | 1.07 | 1.07 |
| Final training loss | 0.23 | 0.23 | 0.22 |
| Final validation loss | 0.33 | 0.27 | 0.27 |
| Max CPU memory per MPI task [GB] | 107.60 | 107.70 | 107.25 |
| MAX GPU memory per MPI task[GB] | 12.04 | 12.04 | 12.04 |
| Node ID | jwc09n066 | jwc09n066 | jwc09n066 |
| Max. GPU power [W] | 310.70 | 296.75 | 297.43 |
| Avg. GPU power [W] | 77.92 | 69.90 | 66.62 |
| GPU energy consumption [Wh] | 33.15 | 29.48 | 28.15 |

Table 5: AP1 JUWELS Cluster training benchmark

| Notes | /scratch_local | /scratch_local | /scratch_local | /data | /data | /data |
|---|---|---|---|---|---|---|
| Experiment number | 0 | 1 | 2 | 0 | 1 | 2 |
| Job ID | 2573 | 2574 | 2575 | 2585 | 2586 | 2587 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 | 32 | 32 | 32 |
| Total runtime | 1610.62 | 1280.85 | 1355.7 | 2095.96 | 1946.28 | 2070.76 |
| Total training time | 1606.51 | 1276.81 | 1351.69 | 2091.96 | 1942.26 | 2066.73 |
| Avg. training time per epoch | 535.42 | 425.53 | 450.49 | 2091.66 | 1942.08 | 2066.4 |
| Performance [GB/s] | 0.62 | 0.77 | 0.73 | 0.16 | 0.17 | 0.16 |
| First epoch training time | 509.67 | 486.1 | 493.15 | 2091.66 | 1942.08 | 2066.4 |
| Min. training time per epoch | 509.67 | 386 | 418.36 | 2091.66 | 1942.08 | 2066.4 |
| Max. training time per epoch | 553.53 | 486.1 | 493.15 | 2091.66 | 1942.08 | 2066.4 |
| Avg. training time per batch | 2.27 | 1.81 | 1.91 | 8.88 | 8.25 | 8.77 |
| Final training loss | 0.217 | 0.219 | 0.207 | 0.466 | 0.5 | 0.46 |
| Final validation loss | 0.279 | 0.197 | 0.209 | 0.231 | 0.32 | 0.208 |
| Max CPU memory per MPI task [GB] | 114.8 | 114.9 | 114.5 | 48.76 | 49.3 | 49 |
| MAX GPU memory per MPI task[GB] | 27 | 27 | 27 | 27 | 27 | 27 |
| Node ID | ICNODE01 | ICNODE01 | ICNODE01 | ICNODE01 | ICNODE01 | ICNODE01 |
| Max. power [W] | 810 | 797 | 806 | 765 | 759 | 781 |
| Avg. power [W] | 583.9683698 | 591.3626374 | 589.0621118 | 527.6299879 | 512.003632 | 514.5656442 |
| Energy consumption [Wh] | 261.26 | 210.40 | 221.83 | 307.19 | 276.81 | 295.98 |
| Action [MJs] | 1514.87047 | 970.1757774 | 1082.650503 | 2317.904033 | 1939.472747 | 2206.481655 |

Table 6: AP1 E4 Intel System training benchmark.

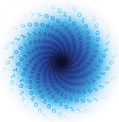


| Notes | /data | /data | /data |
|---|---|---|---|
| Experiments | 3 | 4 | 5 |
| Job ID | 2544 | 2545 | 2565 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime [s] | 106.32 | 96.37 | 111.91 |
| Data loading overhead [s] | 85.62 | 76.04 | 91.06 |
| Total inference time [s] | 20.38 | 20.01 | 20.30 |
| Performance [GB/s] | 0.13 | 0.14 | 0.12 |
| Max CPU memory per MPI task [GB] | 35.62 | 35.66 | 36.55 |
| MAX GPU memory per MPI task[GB] | 5.14 | 5.14 | 5.14 |
| Node ID | ACNODE01 | ACNODE01 | ACNODE01 |
| Max. power [W] | 605.00 | 619.00 | 598.00 |
| Avg. power [W] | 460.45 | 463.28 | 455.36 |
| Energy consumption [Wh] | 10.95 | 9.79 | 11.52 |
| Action [MJs] | 4.191489128 | 3.394900325 | 4.640391006 |

Table 7: AP1 E4 AMD System training benchmark

| Notes | SCRATCH | SCRATCH | SCRATCH |
|---|---|---|---|
| Experiments | 0 | 1 | 2 |
| Job ID | 6753182 | 6753183 | 6753195 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 |
| Total runtime [s] | 40.09 | 44.86 | 40.05 |
| Data loading overhead [s] | 10.56 | 9.67 | 9.42 |
| Total inference time [s] | 29.17 | 34.8 | 30.28 |
| Performance [GB/s] | 0.34 | 0.31 | 0.34 |
| Max CPU memory per MPI task [GB] | 35.17 | 35.26 | 34.75 |
| MAX GPU memory per MPI task[GB] | 7.94 | 7.94 | 7.94 |
| Node ID | jwb0090 | jwb0220 | jwb0090 |
| Max. GPU power [W] | 67.61 | 61.81 | 67.94 |
| Avg. GPU power [W] | 60.78 | 58.39 | 61.00 |
| GPU energy consumption [Wh] | 0.18 Wh | 0.16 Wh | 0.16 Wh |

Table 8: AP1 JUWELS Booster inference benchmark

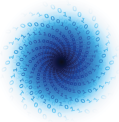


| Notes | /scratch_local | /scratch_local | /scratch_local | /data | /data | /data |
|---|---|---|---|---|---|---|
| Experiments | 3 | 4 | 5 | 6 | 7 | 8 |
| Job ID | 2550 | 2551 | 2567 | 2553 | 2554 | 2566 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 | 32 | 32 | 32 |
| Total runtime [s] | 57.27 | 57.61 | 57.36 | 121.70 | 115.32 | 126.90 |
| Data loading overhead [s] | 22.46 | 22.76 | 22.90 | 86.43 | 80.33 | 91.80 |
| Total inference time [s] | 34.62 | 34.62 | 34.24 | 34.89 | 34.77 | 34.87 |
| Performance [GB/s] | 0.24 | 0.24 | 0.24 | 0.11 | 0.12 | 0.11 |
| Max CPU memory per MPI task [GB] | 34.21 | 34.07 | 34.14 | 34.44 | 34.35 | 34.29 |
| MAX GPU memory per MPI task[GB] | 7.94 | 7.94 | 7.94 | 7.94 | 7.94 | 7.94 |
| Node ID | ICNODE01 | ICNODE01 | ICNODE01 | ICNODE01 | ICNODE01 | ICNODE01 |
| Max. power [W] | 736.00 | 736.00 | 711.00 | 708.00 | 712.00 | 712.00 |
| Avg. power [W] | 533.86 | 532.38 | 510.80 | 511.59 | 507.17 | 505.32 |
| Energy consumption [Wh] | 3.33 | 3.37 | 3.25 | 12.28 | 11.32 | 12.89 |
| Action [MJs] | 0.6866983443 | 0.6980566465 | 0.6709582752 | 5.381200908 | 4.698224133 | 5.886723454 |

Table 9: AP1 E4 Intel System inference benchmark.

| Notes | /data | /data | /data |
|---|---|---|---|
| Experiments | 3 | 4 | 5 |
| Job ID | 2544 | 2545 | 2565 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime [s] | 106.32 | 96.37 | 111.91 |
| Data loading overhead [s] | 85.62 | 76.04 | 91.06 |
| Total inference time [s] | 20.38 | 20.01 | 20.30 |
| Performance [GB/s] | 0.13 | 0.14 | 0.12 |
| Max CPU memory per MPI task [GB] | 35.62 | 35.66 | 36.55 |
| MAX GPU memory per MPI task[GB] | 5.14 | 5.14 | 5.14 |
| Node ID | ACNODE01 | ACNODE01 | ACNODE01 |
| Max. power [W] | 605.00 | 619.00 | 598.00 |
| Avg. power [W] | 460.45 | 463.28 | 455.36 |
| Energy consumption [Wh] | 10.95 | 9.79 | 11.52 |
| Action [MJs] | 4.191489128 | 3.394900325 | 4.640391006 |

Table 10: AP1 E4 AMD System inference benchmark.



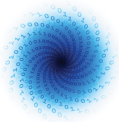
| Experiment number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|
| Job ID | 7460059 | 7460190 | 7460997 | 7437143 | 7437575 | 7453022 | 7438081 | 7450985 | 7451513 | 7451585 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 2 | 2 | 2 | 4 | 4 | 4 | 4 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 |
| Total runtime | 967.259 | 1,066.503 | 1,025.192 | 632.661 | 615.763 | 650.677 | 443.626 | 451.197 | 449.225 | 471.384 |
| Total training time | 930.955 | 1,027.002 | 987.800 | 595.507 | 578.838 | 613.075 | 402.704 | 409.020 | 407.212 | 429.081 |
| Training time for epoch | 893.864 | 990.217 | 950.993 | 560.605 | 544.026 | 576.486 | 366.691 | 372.928 | 369.852 | 392.741 |
| Avg. training time per batch | 0.066 | 0.066 | 0.066 | 0.099 | 0.099 | 0.099 | 0.144 | 0.146 | 0.145 | 0.145 |
| Final training loss | 0.406 | 0.408 | 0.408 | 0.368 | 0.368 | 0.368 | 0.429 | 0.429 | 0.429 | 0.429 |
| Final validation loss | 0.599 | 0.599 | 0.599 | 0.599 | 0.599 | 0.599 | 0.599 | 0.599 | 0.599 | 0.599 |
| Max CPU memory per MPI task [GB] | 3.243 | 3.240 | 3.245 | 3.798 | 3.791 | 3.794 | 4.814 | 4.808 | 4.805 | 4.806 |
| MAX GPU memory per MPI task[GB] | 10.721 | 10.721 | 10.721 | 11.528 | 11.528 | 11.528 | 11.274 | 11.274 | 11.274 | 11.274 |
| Node ID | jwb1159 | jwb0031 | jwb0088 | jwb1067 | jwb0245 | jwb1188 | jwb0281 | jwb0350 | jwb0288 | jwb0858 |
| Max. GPU power [W] | 240.240 | 271.540 | 239.380 | 239.120 | 272.980 | 276.280 | 278.930 | 267.750 | 263.320 | 252.970 |
| Avg. GPU power [W] | 78.350 | 86.460 | 70.240 | 87.320 | 97.170 | 92.100 | 112.140 | 114.730 | 117.610 | 108.910 |
| GPU energy consumption [Wh] | 20.261 | 24.665 | 19.273 | 14.444 | 15.624 | 15.684 | 12.544 | 13.035 | 13.303 | 12.981 |
| Total IO time | 390.333 | 380.952 | 379.683 | 38.688 | 385.153 | 386.987 | 396.006 | 398.073 | 402.729 | 39.226 |

Table 11: AP2 JUWELS Booster training benchmark

| Experiment number | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Job ID | 7458947 | 7459169 | 7459325 | 7457780 | 7458081 | 7458327 | 7451732 | 7452415 | 7452841 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 2 | 2 | 2 | 4 | 4 | 4 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 |
| Total runtime | 1,129.246 | 1,176.583 | 1,171.852 | 906.262 | 842.249 | 832.631 | 583.324 | 566.216 | 642.518 |
| Total training time | 1,095.736 | 1,136.824 | 1,137.338 | 861.277 | 799.024 | 794.623 | 532.549 | 519.437 | 598.011 |
| Training time for epoch | 1,045.277 | 1,086.074 | 1,086.936 | 809.802 | 750.927 | 746.521 | 485.883 | 473.739 | 552.295 |
| Avg. training time per batch | 0.102 | 0.102 | 0.102 | 0.157 | 0.157 | 0.156 | 0.211 | 0.209 | 0.209 |
| Final training loss | 0.401 | 0.401 | 0.401 | 0.405 | 0.405 | 0.405 | 0.418 | 0.417 | 0.418 |
| Final validation loss | 0.600 | 0.600 | 0.600 | 0.599 | 0.599 | 0.599 | 0.600 | 0.600 | 0.600 |
| Max CPU memory per MPI task [GB] | 2.705 | 2.711 | 2.713 | 3.217 | 3.229 | 3.208 | 4.116 | 4.118 | 4.116 |
| MAX GPU memory per MPI task[GB] | 10.989 | 10.989 | 10.989 | 11.501 | 11.501 | 11.501 | 11.643 | 11.643 | 11.643 |
| Node ID | jwc09n039 | jwc09n177 | jwc09n177 | jwc09n039 | jwc09n135 | jwc09n135 | jwc09n180 | jwc09n180 | jwc09n180 |
| Max. GPU power [W] | 234.20 | 232.26 | 242.08 | 263.34 | 269.64 | 262.91 | 242.57 | 254.74 | 251.41 |
| Avg. GPU power [W] | 66.58 | 69.47 | 70.88 | 101.10 | 93.68 | 101.79 | 113.32 | 120.35 | 131.95 |
| GPU energy consumption [Wh] | 20.27 | 21.94 | 22.39 | 24.19 | 20.79 | 22.47 | 16.76 | 17.37 | 21.92 |
| Total IO time | 31.80 | 31.24 | 31.44 | 32.45 | 31.88 | 32.52 | 34.45 | 33.38 | 33.22 |

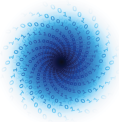
Table 12: AP2 JUWELS Cluster training benchmark

## 6.2 AP 2



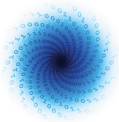
| Notes | i-gpu-a100 | i-gpu-a100 | i-gpu-a100 |
|---|---|---|---|
| Experiment number | 1 | 2 | 3 |
| Job ID | 2388 | 2389 | 2414 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime | 1,026.28 | 1,163.51 | 1,052.30 |
| Total training time | 1,002.64 | 1,139.48 | 1,019.96 |
| Training time for epoch | 948.99 | 1,085.77 | 967.20 |
| Avg. training time per batch | 0.064 | 0.064 | 0.065 |
| Final training loss | 0.408 | 0.408 | 0.407 |
| Final validation loss | 0.599 | 0.599 | 0.599 |
| Max CPU memory per MPI task [GB] | 3.847 | 3.849 | 3.854 |
| MAX GPU memory per MPI task[GB] | 10.721 | 10.721 | 10.721 |
| Node ID | ICNODE01 | ICNODE01 | ICNODE01 |
| Max. GPU power [W] | 606.00 | 595.00 | 611.00 |
| Avg. GPU power [W] | 536.73 | 525.87 | 549.08 |
| Max. GPU power [VA] | 642.00 | 630.00 | 646.00 |
| Avg. GPU power [VA] | 569.22 | 558.93 | 582.86 |
| Node energy consumption [Wh] | 149.49 | 166.45 | 155.57 |
| GPU energy consumption [VAh] | 158.53 | 176.91 | 165.14 |
| Total IO time | 21.71 | 22.71 | 21.03 |
| Action [MJs] | 552.2898937 | 697.1944424 | 589.328428 |

Table 13: AP2 E4 Intel System training benchmark.



| Notes | a-gpu-mi100 | a-gpu-mi100 | a-gpu-mi100 |
|---|---|---|---|
| Experiment number | 1 | 2 | 3 |
| Job ID | 2375 | 2379 | 2387 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime | 1,394.68 | 1,429.90 | 1,423.87 |
| Total training time | 1,364.05 | 1,399.14 | 1,393.22 |
| Training time for epoch | 1,289.41 | 1,324.63 | 1,318.86 |
| Avg. training time per batch | 0.116 | 0.116 | 0.116 |
| Final training loss | 0.383 | 0.383 | 0.383 |
| Final validation loss | 0.599 | 0.599 | 0.599 |
| Max CPU memory per MPI task [GB] | 3.518 | 3.519 | 3.518 |
| MAX GPU memory per MPI task[GB] | 10.998 | 10.998 | 10.998 |
| Node ID | ACNODE02 | ACNODE02 | ACNODE02 |
| Max. GPU power [W] | 606.00 | 604.00 | 604.00 |
| Avg. GPU power [W] | 553.49 | 542.72 | 550.80 |
| Max. GPU power [VA] | 668.00 | 665.00 | 668.00 |
| Avg. GPU power [VA] | 602.15 | 594.87 | 602.60 |
| Node energy consumption [Wh] | 214.43 | 210.93 | 213.16 |
| GPU energy consumption [VAh] | 228.16 | 231.20 | 233.21 |
| Total IO time | 28.49 | 28.79 | 28.52 |
| Action [MJs] | 1076.608036 | 1085.778736 | 1092.656425 |

Table 14: AP2 E4 AMD System training benchmark.

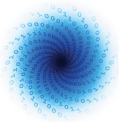


| Notes | | | |
|---|---|---|---|
| Experiments | 0 | 1 | 2 |
| Job ID | 7476664 | 7476699 | 7476739 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 |
| Total runtime [s] | 26.903 | 24.736 | 27.342 |
| Data loading overhead [s] | 7.500 | 5.749 | 6.997 |
| Total inference time [s] | 19.403 | 18.987 | 20.345 |
| Max CPU memory per MPI task [GB] | 2.645 | 2.645 | 2.634 |
| MAX GPU memory per MPI task[GB] | 1.093 | 1.093 | 1.093 |
| Node ID | jwb0033 | jwb0001 | jwb0097 |
| Max. GPU power [W] | 58.060 | 59.090 | 55.910 |
| Avg. GPU power [W] | 56.440 | 56.210 | 54.560 |
| GPU energy consumption [Wh] | 0.422 | 0.386 | 0.414 |

Table 15: AP2 JUWELS Booster inference benchmark

| Experiments | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| Job ID | 7475658 | 7476309 | 7476601 | 7476378 | 7476435 | 7476714 | 7475634 | 7580917 | 7581976 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 2 | 2 | 2 | 4 | 4 | 4 |
| #MPI Tasks | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 |
| Total runtime [s] | 18.850 | 33.124 | 18.953 | 24.951 | 21.797 | 22.120 | 26.927 | 29.074 | 30.365 |
| Data loading overhead [s] | 4.414 | 7.746 | 4.504 | 5.819 | 4.607 | 4.536 | 4.575 | 4.630 | 5.704 |
| Total inference time [s] | 14.436 | 25.378 | 14.450 | 19.132 | 17.190 | 17.583 | 22.352 | 25.232 | 26.371 |
| Max CPU memory per MPI task [GB] | 2.118 | 2.108 | 2.114 | 2.520 | 2.511 | 2.516 | 3.351 | 4.178 | 4.192 |
| MAX GPU memory per MPI task[GB] | 1.093 | 1.093 | 1.093 | 1.093 | 1.093 | 1.093 | 1.093 | 0.980 | 0.980 |
| Node ID | jwc09n174 | jwc09n066 | jwc09n174 | jwc09n066 | jwc09n066 | jwc09n174 | jwc09n174 | jwc09n045 | jwc09n072 |
| Max. GPU power [W] | 42.400 | 41.910 | 175.880 | 41.450 | 41.910 | 42.400 | 94.240 | 45.350 | 43.830 |
| Avg. GPU power [W] | 40.570 | 40.530 | 73.790 | 40.590 | 40.830 | 40.690 | 80.460 | 42.310 | 42.740 |
| GPU energy consumption [Wh] | 0.212 | 0.373 | 0.388 | 0.281 | 0.247 | 0.250 | 0.602 | 0.342 | 0.360 |

Table 16: AP2 JUWELS Cluster inference benchmark

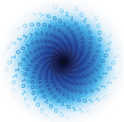


| Notes | i-gpu-a100 | i-gpu-a100 | i-gpu-a100 |
|---|---|---|---|
| Experiments | 0 | 1 | 2 |
| Job ID | 2420 | 2421 | 2422 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime [s] | 16.686 | 14.795 | 14.749 |
| Data loading overhead [s] | 3.982 | 3.712 | 3.715 |
| Total inference time [s] | 12.704 | 11.082 | 11.034 |
| Max CPU memory per MPI task [GB] | 3.271 | 3.271 | 3.274 |
| MAX GPU memory per MPI task[GB] | 1.093 | 1.093 | 1.093 |
| Node ID | ICNODE01 | ICNODE01 | ICNODE01 |
| Max. GPU power [W] | 655.000 | 650.000 | 654.000 |
| Avg. GPU power [W] | 480.020 | 521.560 | 529.580 |
| Max. GPU power [VA] | 672.000 | 667.000 | 672.000 |
| Avg. GPU power [VA] | 498.14 | 539.36 | 547.50 |
| GPU energy consumption [Wh] | 2.22 | 2.14 | 2.17 |
| GPU energy consumption [VAh] | 2.31 | 2.22 | 2.24 |
| Action [MJs] | 0.1336452107 | 0.1141576082 | 0.1151995645 |

Table 17: AP2 E4 Intel System inference benchmark

| Notes | a-gpu-mi100 | a-gpu-mi100 | a-gpu-mi100 |
|---|---|---|---|
| Experiments | 0 | 1 | 2 |
| Job ID | 2423 | 2424 | |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime [s] | 29.029 | 18.723 | 18.751 |
| Data loading overhead [s] | 4.601 | 3.600 | 3.606 |
| Total inference time [s] | 24.428 | 15.123 | 15.145 |
| Max CPU memory per MPI task [GB] | 2.962 | 2.966 | 2.965 |
| MAX GPU memory per MPI task[GB] | 1.093 | 1.093 | 1.093 |
| Node ID | ACNODE02 | ACNODE02 | ACNODE02 |
| Max. GPU power [W] | 639.000 | 637.000 | 634.000 |
| Avg. GPU power [W] | 446.490 | 469.690 | 476.800 |
| Max. GPU power [VA] | 657.000 | 655.000 | 650.000 |
| Avg. GPU power [VA] | 473.18 | 495.17 | 502.00 |
| GPU energy consumption [Wh] | 3.60 | 2.44 | 2.48 |
| GPU energy consumption [VAh] | 3.82 | 2.58 | 2.61 |
| Action [MJs] | 0.3762442772 | 0.1646501719 | 0.1676410924 |

Table 18: AP2 E4 AMD System inference benchmark



## 6.3 AP 3

| Data location | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | CSCRATCH | CSCRATCH | CSCRATCH |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Experiment flags | –nocache | –nocache | –nocache | | | | –dl_test –nocache | –dl_test –nocache | –dl_test –nocache | –nocache | –nocache | –nocache |
| Experiment number | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| Job ID | 6741790 | 6741813 | 6741825 | 6741726 | 6741727 | 6741789 | 6746363 | 6746364 | 6746365 | 6746376 | 6746377 | 6746378 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 | 48 |
| Total runtime [s] | 1617.56 | 1616.99 | 1616.9 | 1199.76 | 1137.32 | 1137.8 | 1012.75 | 1012.63 | 1012.31 | 1918.86 | 1918.51 | 1919.04 |
| Total training time [s] | 1614.8 | 1614.84 | 1614.79 | 1195.21 | 1135.12 | 1135.14 | 1009.7 | 1009.72 | 1009.7 | 1916.53 | 1916.25 | 1916.25 |
| Avg. training time per epoch [s] | 284.2 | 285.2 | 294.2 | 185.2 | 181.6 | 182.4 | 172.4 | 173.4 | 172.6 | 338 | 333.4 | 333.2 |
| Performance [GB/s] | 0.22 | 0.22 | 0.22 | 0.29 | 0.31 | 0.31 | 0.35 | 0.35 | 0.35 | 0.18 | 0.18 | 0.18 |
| First epoch training time [s] | 295 | 282 | 307 | 333 | 325 | 323 | 180 | 181 | 179 | 355 | 347 | 346 |
| Min. training time per epoch | 278 | 281 | 289 | 148 | 145 | 147 | 169 | 170 | 170 | 333 | 330 | 330 |
| Max. training time per epoch | 295 | 293 | 307 | 333 | 325 | 323 | 180 | 181 | 179 | 355 | 347 | 346 |
| Avg. training time per batch | 0.06 | 0.06 | 0.06 | 0.04 | 0.04 | 0.04 | 0.03 | 0.03 | 0.03 | 0.07 | 0.07 | 0.07 |
| Final training loss | 0.0335 | 0.0366 | 0.0131 | 0.0352 | 0.0362 | 0.0092 | | | | 0.041 | 0.0447 | 0.0395 |
| Final validation loss | 0.05 | 0.0534 | 0.022 | 0.049 | 0.0522 | 0.0206 | | | | 0.0527 | 0.0603 | 0.0668 |
| Max CPU memory per MPI task [GB] | 4.47 | 4.44 | 4.46 | 65.4 | 65.4 | 65.45 | 1.8 | 1.8 | 1.8 | 4.46 | 4.49 | 4.43 |
| MAX GPU memory per MPI task[GB] | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0.5 | 0 | 0 | 0 | 0.61 | 0.61 | 0.61 |
| Node ID (job report) | jwb0065 | jwb0021 | jwb0033 | jwb0021 | jwb0033 | jwb0053 | jwb0193 | jwb0023 | jwb0962 | jwb0023 | jwb0193 | jwb0907 |
| Max. GPU power (job report) | 113.57 | 124.27 | 103.07 | 131.22 | 136.48 | 132.64 | 65.59 | 62.46 | 80.6 | 139.03 | 139.28 | 140.9 |
| Avg. GPU power (job report) | 64.89 | 65.07 | 62.45 | 68.01 | 66.8 | 68.53 | 58.96 | 59.83 | 57.75 | 69.99 | 69.51 | 68.27 |
| GPU energy consumption [Wh] | 29.156519 | 29.22709425 | 28.04872361 | 22.665466 | 21.10360444 | 21.65928722 | 16.58659444 | 16.82934803 | 16.23913958 | 37.3058365 | 37.04323058 | 36.39246133 |

Table 19: AP3 JUWELS Booster training benchmark

| Data location | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH | SCRATCH |
|---|---|---|---|---|---|---|---|---|---|
| Experiment flags | –nocache | –nocache | –nocache | | | | –dl_test –nocache | –dl_test –nocache | –dl_test –nocache |
| Experiment number | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| Job ID | 6753256 | 6753257 | 6753259 | 6753260 | 6753262 | 6753264 | 6753266 | 6753267 | 6753268 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 | 40 |
| Total runtime [s] | 2218.25 | 2218.07 | 2218.19 | 1498.14 | 1498.55 | 1498.38 | 1492.45 | 1492.31 | 1612.33 |
| Total training time [s] | 2215.76 | 2215.59 | 2215.62 | 1495.58 | 1495.59 | 1495.74 | 1489.69 | 1489.71 | 1609.71 |
| Avg. training time per epoch [s] | 407.2 | 405.6 | 406 | 279.8 | 280.4 | 280.8 | 283.8 | 268.2 | 286 |
| Performance [GB/s] | 0.16 | 0.16 | 0.16 | 0.23 | 0.23 | 0.23 | 0.23 | 0.23 | 0.22 |
| First epoch training time [s] | 436 | 432 | 434 | 425 | 431 | 433 | 351 | 288 | 283 |
| Min. training time per epoch | 399 | 397 | 398 | 242 | 242 | 241 | 253 | 258 | 268 |
| Max. training time per epoch | 436 | 432 | 434 | 425 | 431 | 433 | 351 | 288 | 306 |
| Avg. training time per batch | 0.08 | 0.08 | 0.08 | 0.05 | 0.05 | 0.05 | 0.05 | 0.05 | 0.06 |
| Final training loss | 0.0369 | 0.0402 | 0.0445 | 0.0377 | 0.042 | 0.0463 | | | |
| Final validation loss | 0.0583 | 0.0578 | 0.056 | 0.0566 | 0.0535 | 0.0601 | | | |
| Max CPU memory per MPI task [GB] | 3.59 | 3.56 | 3.54 | 64.45 | 64.48 | 64.44 | 1.83 | 1.79 | 1.79 |
| MAX GPU memory per MPI task[GB] | 0.63 | 0.63 | 0.61 | 0.63 | 0.63 | 0.61 | 0 | 0 | 0 |
| Node ID (job report) | jwc09n000 | jwc09n006 | jwc09n000 | jwc09n006 | jwc09n003 | jwc09n000 | jwc09n003 | jwc09n000 | jwc09n003 |
| Max. GPU power (job report) | 148.06 | 149.03 | 137.87 | 145.83 | 146.6 | 141.47 | 59.42 | 57.96 | 58.45 |
| Avg. GPU power (job report) | 56.53 | 56.83 | 55.71 | 58.73 | 57.62 | 57.09 | 45.99 | 44.93 | 45.87 |
| GPU energy consumption [Wh] | 34.83268681 | 35.01469947 | 34.32649025 | 24.4404895 | 23.98512528 | 23.7618095 | 19.06604875 | 18.62485786 | 20.54377142 |

Table 20: AP3 JUWELS Cluster training benchmark

| Data location | /data | /data | /data | /data | /data | /data | /data | /data | /data | scratch_local | scratch_local | scratch_local |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| Experiment flags | –nocache | –nocache | –nocache | | | | –dl_test –nocache | –dl_test –nocache | –dl_test –nocache | –nocache | –nocache | –nocache |
| Experiment number | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| Job ID | 2333 | 2334 | 2335 | 2336 | 2337 | 2338 | 2339 | 2340 | 2341 | 2356 | 2357 | 2358 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| Total runtime [s] | 803.23 | 792.08 | 786.48 | 877.62 | 867.62 | 878.27 | 598.2 | 618.75 | 593.23 | 803.71 | 797.69 | 796.13 |
| Total training time [s] | 801.82 | 790.48 | 785.06 | 876.21 | 866.21 | 876.83 | 596.75 | 617.31 | 591.8 | 802.19 | 796.16 | 794.58 |
| Avg. training time per epoch [s] | 160 | 158.2 | 156.8 | 175 | 173.2 | 175.4 | 119.2 | 123.6 | 118.2 | 160.4 | 159.2 | 158.8 |
| Performance [GB/s] | 0.44 | 0.44 | 0.45 | 0.4 | 0.4 | 0.4 | 0.59 | 0.57 | 0.59 | 0.44 | 0.44 | 0.44 |
| First epoch training time [s] | 164 | 164 | 162 | 294 | 292 | 293 | 119 | 122 | 118 | 165 | 164 | 163 |
| Min. training time per epoch | 158 | 156 | 151 | 145 | 143 | 146 | 119 | 122 | 118 | 158 | 157 | 157 |
| Max. training time per epoch | 164 | 164 | 162 | 294 | 292 | 293 | 120 | 126 | 119 | 165 | 164 | 163 |
| Avg. training time per batch | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.03 | 0.02 | 0.02 | 0.02 | 0.03 | 0.03 | 0.03 |
| Final training loss | 0.0349 | 0.0425 | 0.0404 | 0.0412 | 0.0357 | 0.0268 | | | | 0.0318 | 0.0394 | 0.0447 |
| Final validation loss | 0.0524 | 0.056 | 0.0528 | 0.0577 | 0.0653 | 0.0435 | | | | 0.0566 | 0.066 | 0.0677 |
| Max CPU memory per MPI task [GB] | 6.36 | 6.23 | 6.91 | 65.42 | 65.41 | 65.46 | 3.81 | 3.81 | 3.83 | 6.08 | 6.29 | 6.47 |
| MAX GPU memory per MPI task[GB] | 0.63 | 0.63 | 0.62 | 0.61 | 0.62 | 0.62 | 0 | 0 | 0 | 0.62 | 0.64 | 0.62 |
| Node ID (job report) | ICNODE01 | ICNODE02 | ICNODE02 | ICNODE01 | ICNODE02 | ICNODE01 | ICNODE02 | ICNODE01 | ICNODE02 | ICNODE01 | ICNODE02 | ICNODE02 |
| Max. node power [W] | 623 | 972 | 969 | 634 | 937 | 608 | 687 | 549 | 690 | 633 | 773 | 772 |
| Avg. node power [W] | 606.058104 | 911.0756173 | 911.7794793 | 589.9580514 | 816.3465753 | 588.1300813 | 674.2429719 | 531.8774319 | 676.0080321 | 605.9926036 | 746.0534918 | 745.9142012 |
| Node energy consumption [Wh] | 135.2233475 | 200.4568819 | 199.1934236 | 143.8219403 | 196.7440599 | 143.4825018 | 112.0367072 | 91.41643361 | 111.3967347 | 135.2895321 | 165.3109472 | 164.9568536 |
| Action [MJs] | 391.0156178 | 571.6003933 | 563.9819176 | 454.3956405 | 614.5166926 | 453.6589567 | 241.2732896 | 203.6301059 | 237.9019857 | 391.4407793 | 474.7208021 | 472.7775595 |

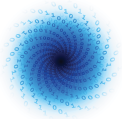
Table 21: AP3 E4 Intel System training benchmark

| Data location | /data | /data | /data | /data | /data | /data | /data | /data | /data |
|---|---|---|---|---|---|---|---|---|---|
| Experiment flags | –nocache | –nocache | –nocache | | | | –dl_test –nocache | –dl_test –nocache | –dl_test –nocache |
| Experiment number | 0 | 1 | 2 | 0 | 1 | 2 | 0 | 1 | 2 |
| Job ID | 2342 | 2515 | 2344 | 2345 | 2346 | 2347 | 2348 | 2349 | 2350 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 | 32 |
| Total runtime [s] | 2213.33 | 2716.87 | 2215.39 | 2227.8 | 1936.22 | 1920.46 | 236.54 | 234.48 | 234.24 |
| Total training time [s] | 2209.45 | 2713.69 | 2210.08 | 2222.51 | 1934.95 | 1919.2 | 235.25 | 233.18 | 232.94 |
| Avg. training time per epoch [s] | 441.8 | 542.6 | 442.2 | 444.4 | 387 | 383.8 | 47.2 | 46.6 | 46.4 |
| Performance [GB/s] | 0.16 | 0.13 | 0.16 | 0.16 | 0.18 | 0.18 | 1.49 | 1.5 | 1.5 |
| First epoch training time [s] | 697 | 1191 | 695 | 695 | 393 | 389 | 48 | 47 | 47 |
| Min. training time per epoch | 377 | 380 | 379 | 381 | 385 | 381 | 47 | 46 | 46 |
| Max. training time per epoch | 697 | 1191 | 695 | 695 | 393 | 389 | 48 | 47 | 47 |
| Avg. training time per batch | 0.08 | 0.09 | 0.08 | 0.08 | 0.07 | 0.07 | 0.01 | 0.01 | 0.01 |
| Final training loss | 0.04 | 0.0444 | 0.0372 | 0.0406 | 0.0412 | 0.0328 | | | |
| Final validation loss | 0.0561 | 0.0586 | 0.0512 | 0.0613 | 0.0587 | 0.0498 | | | |
| Max CPU memory per MPI task [GB] | 4.94 | 6.32 | 4.91 | 65.16 | 64.96 | 66.04 | 2.4 | 2.56 | 2.48 |
| MAX GPU memory per MPI task[GB] | 0.77 | 0.77 | 0.75 | 0.76 | 0.75 | 0.75 | 0 | 0 | 0 |
| Node ID (job report) | ACNODE02 | ACNODE02 | ACNODE01 | ACNODE01 | ACNODE02 | ACNODE01 | ACNODE01 | ACNODE01 | ACNODE01 |
| Max. node power [W] | 690 | 698 | 689 | 622 | 653 | 639 | 649 | 686 | 639 |
| Avg. node power [W] | 607.8359133 | 513.3514019 | 601.1093943 | 504.6666667 | 565.035409 | 607.3316327 | 610.8622449 | 610.5606061 | 607.3316327 |
| Node energy consumption [Wh] | 373.7059617 | 387.4191731 | 369.9143725 | 312.3045556 | 303.8980166 | 323.9878076 | 40.13704317 | 39.76784747 | 39.5170449 |
| Rest time [s] | 3.88 | 3.18 | 5.31 | 5.29 | 1.27 | 1.26 | 1.29 | 1.3 | 1.3 |
| Action [MJs] | 2977.684618 | 3789.243104 | 2950.216566 | 2504.70752 | 2118.288304 | 2239.94025 | 34.17845829 | 33.56915355 | 33.32330135 |

Table 22: AP3 E4 AMD System training benchmark

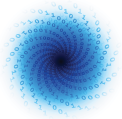
| Experiments | 0 | 1 | 2 |
|---|---|---|---|
| Job ID | 6750557 | 6750558 | 6750559 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 |
| Total runtime [s] | 44.58 | 44.62 | 44.5 |
| Data loading overhead [s] | 1.12 | 1.13 | 1.04 |
| Total inference time [s] | 42.25 | 42.27 | 42.25 |
| Performance [GB/s] | 1.66 | 1.66 | 1.66 |
| Max CPU memory per MPI task [GB] | 3.78 | 3.77 | 3.77 |
| MAX GPU memory per MPI task[GB] | 0.3 | 0.3 | 0.3 |

Table 23: AP3 JUWELS Booster inference benchmark



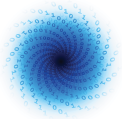
| Experiments | 0 | 1 | 2 |
|---|---|---|---|
| Job ID | 6753236 | 6753241 | 6753243 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 40 | 40 | 40 |
| Total runtime [s] | 44.3 | 44.63 | 44.24 |
| Data loading overhead [s] | 1.08 | 1.44 | 1.01 |
| Total inference time [s] | 42.05 | 42.04 | 42.05 |
| Performance [GB/s] | 1.66 | 1.67 | 1.66 |
| Max CPU memory per MPI task [GB] | 2.9 | 2.9 | 2.92 |
| MAX GPU memory per MPI task[GB] | 0.3 | 0.3 | 0.3 |

Table 24: AP3 JUWELS Cluster inference benchmark



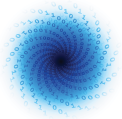
| Experiments | 0 | 1 | 2 |
|---|---|---|---|
| Job ID | 1674 | 1675 | 1676 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime [s] | 34.36 | 30.99 | 28.41 |
| Data loading overhead [s] | 7.56 | 1.27 | 0.79 |
| Total inference time [s] | 25.62 | 28.83 | 26.91 |
| Performance [GB/s] | 2.73 | 2.43 | 2.6 |
| Max CPU memory per MPI task [GB] | 3.8 | 3.81 | 3.8 |
| MAX GPU memory per MPI task[GB] | 0.31 | 0.31 | 0.31 |

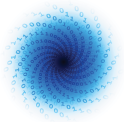
Table 25: AP3 E4 Intel System inference benchmark



| Experiments | 0 | 1 | 2 |
|---|---|---|---|
| Job ID | 1677 | 1678 | 1679 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 32 | 32 | 32 |
| Total runtime [s] | 15.18 | 15.25 | 15.49 |
| Data loading overhead [s] | 0.63 | 0.61 | 0.66 |
| Total inference time [s] | 13.15 | 13.24 | 13.69 |
| Performance [GB/s] | 5.32 | 5.29 | 5.11 |
| Max CPU memory per MPI task [GB] | 3.29 | 3.33 | 3.3 |
| MAX GPU memory per MPI task[GB] | 0.33 | 0.33 | 0.33 |

Table 26: AP3 E4 AMD System inference benchmark

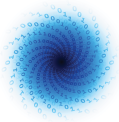




| Experiment number | Booster-SCRATCH | Booster-SCRATCH | Booster-SCRATCH | Booster-CSCRATCH | Booster-CSCRATCH | Booster-CSCRATCH |
|---|---|---|---|---|---|---|
| Job ID | 6748650 | 6748654 | 6748658 | 6752412 | 6752591 | 6752592 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| Loading data time | 3,141.000 | 3,112.000 | 3,033.000 | 17433 | 16785 | 17,000.000 |
| Total training time [s] | 75.000 | 73.000 | 74.000 | 90.000 | 75.000 | 763.000 |
| Total runtime [s] | 3,305.000 | 3,260.000 | 3,158.000 | 17680 | 17010 | 17,914.000 |
| Total training time | 3,216.000 | 3,185.000 | 3,107.000 | 17523 | 16860 | 17,763.000 |
| Avg. training time per epoch [s] | 1,072.000 | 1,061.000 | 1,035.000 | 5841 | 5620 | 5,921.000 |
| First epoch training time [s] | 1,159.000 | 1,060.200 | 1,020.000 | 6238 | 5582 | 6,245.000 |
| Min. training time per epoch | 1,023.000 | 1,033.000 | 1,020.000 | 5618 | 5653 | 5,713.000 |
| Max. training time per epoch | 1,159.000 | 1,092.000 | 1,044.000 | 6238 | 5624 | 6,245.000 |
| Avg. training time per iteration | 5.60E-01 | 5.50E-01 | 5.40E-01 | 3.08 | 2.97 | 3.10E+00 |
| Final training loss | 4.90E-05 | 1.10E-05 | 1.30E-06 | 1.50E-05 | 3.50E-06 | 2.40E-06 |
| Final validation loss | 2.38E-06 | 5.00E-06 | 3.40E-07 | 3.20E-07 | 2.10E-06 | 9.60E-06 |
| Saving model time | 0.250 | 0.130 | 0.050 | 0.04 | 0.04 | 0.040 |
| Max. GPU power | 94.13 | 98.21 | 100.38 | 124.90 | 113.10 | 101.70 |
| Avg. GPU Power | 60.32 | 65.10 | 71.12 | 58.53 | 61.20 | 56.10 |
| GPU Energy consumption [Wh] | 55.37711111 | 58.95166667 | 62.38804444 | 287.4473333 | 289.17 | 279.1598333 |

Table 27: AP4 JUWELS Booster training benchmark

## 6.4 AP 4

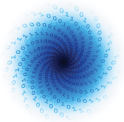


| Experiment number | SCRATCH | SCRATCH | SCRATCH | CSCRATCH | CSCRATCH | CSCRATCH |
|---|---|---|---|---|---|---|
| Job ID | 6754748 | 6754749 | 6754750 | 6762119 | 6762120 | 6762121 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| Loading data time | 7,594.000 | 7,731.000 | 7,580.000 | 20,005.000 | 19,999.000 | 19,680.000 |
| Total training time [s] | 84.000 | 98.000 | 84.000 | 81.000 | 82.000 | 80.000 |
| Total runtime | 7,818.000 | 7,967.000 | 7,814.000 | 20,314.000 | 20,308.000 | 19,971.000 |
| Total training time | 7,678.000 | 7,829.000 | 7,664.000 | 20,086.000 | 20,081.000 | 19,760.000 |
| Avg. training time per epoch | 2,559.300 | 2,609.000 | 2,554.000 | 6,695.333 | 6,693.667 | 6,586.667 |
| First epoch training time | 2,559.000 | 2,689.000 | 2,559.000 | 6,712.000 | 6,685.000 | 6,585.000 |
| Min. training time per epoch | 2,430.000 | 2,409.000 | 2,455.000 | 6,668.000 | 6,685.000 | 6,585.000 |
| Max. training time per epoch | 2,689.000 | 2,731.000 | 2,649.000 | 6,712.000 | 6,698.000 | 6,588.000 |
| Avg. training time per iteration | 1.30E+00 | 1.33E+00 | 1.30E+00 | 3.54E+00 | 3.54E+00 | 3.48E+00 |
| Final training loss | 3.50E-06 | 3.40E-06 | 3.46E-06 | 1.60E-06 | 4.90E-06 | 9.30E-06 |
| Final validation loss | 2.00E-04 | 1.40E-06 | 1.18E-06 | 6.70E-06 | 4.30E-04 | 3.80E-06 |
| Saving model time | 0.830 | 0.220 | 0.750 | 0.030 | 0.070 | 0.100 |

Table 28: AP4 JUWELS Cluster training benchmark

| Experiment number | 1 | 2 | 3 |
|---|---|---|---|
| Job ID | 2384 | 2385 | 2386 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 |
| #CPUs | 16 | 16 | 16 |
| Loading data time | 8,686.000 | 6,794.000 | 6,582.000 |
| Total training time [s] | 68.000 | 68.000 | 59.000 |
| Total runtime | 8914 | 7,018.000 | 6,754.000 |
| Total training time | 8,754.000 | 6,862.000 | 6,641.000 |
| Avg. training time per epoch | 2,918.000 | 2,287.333 | 2,213.667 |
| First epoch training time | 3,215.000 | 2,240.000 | 2,207.000 |
| Min. training time per epoch | 2,769.000 | 2,240.000 | 2,207.000 |
| Max. training time per epoch | 3,215.000 | 2,348.000 | 2,237.000 |
| Avg. training time per iteration | 1.50E+00 | 1.20E+00 | 1.16E+00 |
| Final training loss | 3.80E-05 | 4.20E-06 | 2.40E-06 |
| Final validation loss | 4.30E-06 | 1.50E-06 | 8.90E-06 |
| Saving model time | 0.110 | 0.120 | 0.030 |

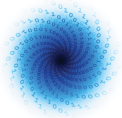
Table 29: AP4 E4 Intel System training benchmark



## 6.5 AP 5

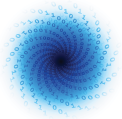
| Experiment number | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Data location | SCRATCH | SCRATCH | SCRATCH | CSCRATCH | CSCRATCH | CSCRATCH |
| Job ID | 7416227 | 7416228 | 7415230 | 7416237 | 7416238 | 7416242 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs | 48 | 48 | 48 | 48 | 48 | 48 |
| Loading data time [s] | 1,945.4 | 1,960.5 | 1,941.1 | 2,318.2 | 2,400.4 | 2,403.9 |
| Total runtime [s] | 2,913.7 | 2,881.4 | 2,932.2 | 2,822.1 | 2,845.8 | 2,819.7 |
| Total training time [s] | 2,882.4 | 2,857.4 | 2,906.6 | 2,773.2 | 2,805.8 | 2,778.0 |
| Avg. training time per epoch [s] | 576.48 | 571.47 | 581.32 | 554.63 | 561.16 | 555.61 |
| Data throughput [MB/s] | 505.38 | 507.09 | 501.79 | 520.24 | 520.23 | 522.23 |
| First epoch training time [s] | 616 | 600 | 611 | 592 | 597 | 590 |
| Min. training time per epoch | 550 | 554 | 559 | 539 | 541 | 538 |
| Max. training time per epoch | 616 | 600 | 611 | 592 | 597 | 590 |
| Avg. training time per iteration | 0.250 | 0.244 | 0.248 | 0.241 | 0.243 | 0.240 |
| Final training loss | 1.29E-01 | 1.37E-01 | 1.36E-01 | 1.38E-01 | 1.72E-01 | 2.06E-01 |
| Final validation loss | 1.39E-01 | 1.20E-01 | 1.24E-01 | 1.20E-01 | 1.29E-01 | 1.29E-01 |
| Saving model time | 7.48 | 6.08 | 5.78 | 5.76 | 5.86 | 5.63 |
| Node ID | jwb0117 | jwb0129 | jwb0149 | jwb0097 | jwb0021 | jwb0117 |
| Max CPU memory per MPI task [GB] | 87,66 | 89.24 | 83.04 | 84.26 | 85.97 | 84.59 |
| MAX GPU memory per MPI task[GB] | 2,97 | 2,90 | 2,97 | 2.93 | 3.01 | 3.00 |
| Max. GPU power | 325.00 | 329.70 | 298.40 | 322.00 | 318.20 | 322.60 |
| Avg. GPU power [W] | 106.30 | 107.50 | 97.10 | 102.80 | 105.40 | 106.20 |
| GPU energy consumption | 86.74 | 86.43 | 79.23 | 80.29 | 84.32 | 83.26 |

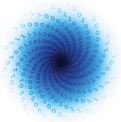
Table 30: AP5 JUWELS Booster training benchmark



| Experiment number | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Data location | SCRATCH | SCRATCH | SCRATCH | CSCRATCH | CSCRATCH | CSCRATCH |
| Job ID | 7416195 | 7416225 | 7417653 | 7416231 | 7416233 | 7416236 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs | 48 | 48 | 48 | 48 | 48 | 48 |
| Loading data time [s] | 3,627.2 | 3,575.3 | 3,568.7 | 3,719.2 | 3,687.8 | 3,623.7 |
| Total runtime [s] | 5,363.8 | 5,343.5 | 5,368.3 | 5,362.2 | 5,282.4 | 5,274.1 |
| Total training time [s] | 5,299.6 | 5,280.9 | 5,300.3 | 5,308.9 | 5,221.9 | 5,205.0 |
| Avg. training time per epoch [s] | 1,059.92 | 1,056.18 | 1,060.05 | 1,061.78 | 1,044.37 | 1,041.00 |
| Performance [GB/s] | 272.73 | 272.08 | 271.54 | 272.20 | 275.21 | 277.71 |
| First epoch training time [s] | 1121 | 1111 | 1112 | 1112 | 1091 | 1097 |
| Min. training time per epoch | 1036 | 1022 | 1034 | 1030 | 1016 | 1016 |
| Max. training time per epoch | 1121 | 1111 | 1112 | 1112 | 1091 | 1097 |
| Avg. training time per batch | 0.456 | 0.452 | 0.452 | 0.452 | 0.443 | 0.446 |
| Final training loss | 1.34E-01 | 1.34E-01 | 1.35E-01 | 1.51E-01 | 1.32E-01 | 1.53E-01 |
| Final validation loss | 1.24E-01 | 1.28E-01 | 1.24E-01 | 1.20E-01 | 1.18E-01 | 1.33E-01 |
| Saving model time | 11.44 | 12.31 | 12.31 | 12.39 | 12.33 | 11.55 |
| Max CPU memory per MPI task [GB] | 85.91 | 82.07 | 83.64 | 81,73 | 84.33 | 84.83 |
| MAX GPU memory per MPI task[GB] | 3.21 | 3.14 | 3.16 | 3.19 | 3.32 | 3.29 |
| Node ID | jwc09n024 | jwc09n024 | jwc09n024 | jwc09n000 | jwc09n024 | jwc09n027 |
| Max. GPU power [W] | 300.40 | 303.10 | 304.70 | 313.20 | 303.30 | 302.40 |
| Avg. GPU power [W] | 87.70 | 86.90 | 87.70 | 86.00 | 86.50 | 92.10 |
| GPU energy consumption [Wh] | 129.53 | 130.87 | 132.06 | 129.00 | 128.71 | 136.12 |

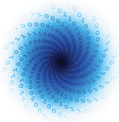
Table 31: AP5 JUWELS Cluster training benchmark





| Experiment number | 1 | 2 | 3 |
|---|---|---|---|
| Data location | /data | /data | /data |
| Job ID | 2087 | 2088 | 2092 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 |
| #CPUs | 48 | 48 | 48 |
| Loading data time [s] | 2,603.3 | 3,037.9 | 2,613.4 |
| Total runtime [s] | 3,017.2 | 3,415.5 | 3,098.0 |
| Total training time [s] | 2,993.9 | 3,377.2 | 3,075.5 |
| Avg. training time per epoch [s] | 598.78 | 675.44 | 615.10 |
| Performance [GB/s] | 481.79 | 429.38 | 470.91 |
| First epoch training time [s] | 621 | 702 | 632 |
| Min. training time per epoch | 582 | 651 | 601 |
| Max. training time per epoch | 621 | 702 | 632 |
| Avg. training time per batch | 0.252 | 0.285 | 0.257 |
| Final training loss | 1.61E-01 | 1.65E-01 | 1.43E-01 |
| Final validation loss | 1.21E-01 | 1.37E-01 | 1.23E-01 |
| Saving model time | 4.89 | 9.09 | 4.69 |
| Max CPU memory per MPI task [GB] | 84.750 | 84.890 | 84.220 |
| MAX GPU memory per MPI task[GB] | 3.16 | 3.30 | 3.21 |
| Node ID | icnode01 | icnode02 | icnode01 |
| Max. GPU power [W] | 1,163.00 | 1,252.00 | 1,166.00 |
| Avg. GPU power [W] | 857.05 | 967.53 | 860.92 |
| GPU energy consumption [Wh] | 722.54 | 922.92 | 743.50 |
| Action [MJs] | 7848.171677 | 11348.03974 | 8292.1068 |

Table 32: AP5 E4 Intel System training benchmark

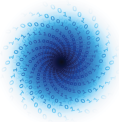


| Experiment number | 1 | 2 | 3 |
|---|---|---|---|
| Data location | /data | /data | /data |
| Job ID | 2089 | 2090 | 2091 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI tasks | 1 | 1 | 1 |
| #CPUs | 48 | 48 | 48 |
| Loading data time [s] | 1,432.9 | 1,431.8 | 1,457.6 |
| Total runtime [s] | 5,003.7 | 4,993.5 | 4,990.7 |
| Total training time [s] | 4,983.5 | 4,968.2 | 4,935.4 |
| Avg. training time per epoch [s] | 996.70 | 993.64 | 987.08 |
| Performance [GB/s] | 289.90 | 290.65 | 292.19 |
| First epoch training time [s] | 1030 | 1029 | 1020 |
| Min. training time per epoch | 985 | 977 | 977 |
| Max. training time per epoch | 1030 | 1029 | 1020 |
| Avg. training time per batch | 0.419 | 0.418 | 0.415 |
| Final training loss | 1.29E-01 | 2.04E-01 | 1.21E-01 |
| Final validation loss | 1.19E-01 | 1.20E-01 | 1.18E-01 |
| Saving model time | 4.70 | 9.53 | 4.65 |
| Max CPU memory per MPI task [GB] | 90.750 | 91.030 | 90.390 |
| MAX GPU memory per MPI task[GB] | 2.72 | 2.67 | 2.63 |
| Node ID | acnode02 | acnode01 | acnode01 |
| Max. GPU power [W] | 866.00 | 852.00 | 864.00 |
| Avg. GPU power [W] | 735.06 | 732.18 | 731.67 |
| GPU energy consumption [Wh] | 1,027.45 | 1,019.77 | 1,018.24 |
| Action [MJs] | 18,507.79 | 18,332.00 | 18,294.23 |

Table 33: AP5 E4 AMD System training benchmark

| Experiment number | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Data location | SCRATCH | SCRATCH | SCRATCH | CSCRATCH | CSCRATCH | CSCRATCH |
| Job ID | 7435787 | 7435788 | 7435789 | 7436875 | 7436876 | 7436879 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 | 48 | 48 | 48 |
| Total runtime [s] | 31.20 | 22.47 | 22.18 | 40.13 | 31.92 | 31.77 |
| Model loading [s] | 3.85 | 2.87 | 2.92 | 3.81 | 2.71 | 2.91 |
| Data loading [s] | 9.41 | 9.20 | 9.26 | 19.75 | 20.02 | 18.78 |
| Total inference time [s] | 17.87 | 10.37 | 9.96 | 16.48 | 9.12 | 10.04 |
| Performance [GB/s] | 0.24 | 0.41 | 0.42 | 0.25 | 0.46 | 0.42 |
| Max CPU memory per MPI task [GB] | 36.05 | 36.09 | 36.08 | 36.09 | 36.08 | 36.1 |
| MAX GPU memory per MPI task[GB] | 5.52 | 5.52 | 5.52 | 5.52 | 5.52 | 5.52 |
| Node ID | jwb0097 | jwb0033 | jwb0053 | jwb0149 | jwb0149 | jwb0021 |

Table 34: AP5 JUWELS Booster inference benchmark

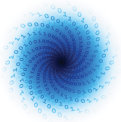


| Experiment number | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Data location | SCRATCH | SCRATCH | SCRATCH | CSCRATCH | CSCRATCH | CSCRATCH |
| Job ID | 7435754 | 7435780 | 7435786 | 7436871 | 7436874 | 7437653 |
| #Nodes | 1 | 1 | 1 | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 | 1 | 1 | 1 |
| #CPUs per task | 48 | 48 | 48 | 48 | 48 | 48 |
| Total runtime [s] | 65.94 | 56.24 | 50.63 | 57.39 | 56.91 | 56.80 |
| Model loading [s] | 6.79 | 7.41 | 7.16 | 7.62 | 7.05 | 7.35 |
| Data loading [s] | 43.12 | 33.24 | 27.43 | 32.60 | 34.22 | 32.75 |
| Total inference time [s] | 16.02 | 15.58 | 16.00 | 17.08 | 15.59 | 15.74 |
| Performance [GB/s] | 0.26 | 0.27 | 0.26 | 0.25 | 0.27 | 0.27 |
| Max CPU memory per MPI task [GB] | 35.02 | 35.01 | 35.01 | 35.01 | 35.01 | 35.01 |
| MAX GPU memory per MPI task[GB] | 5.61 | 5.61 | 5.61 | 5.61 | 5.61 | 5.61 |
| Node ID | jwc09n024 | jwc09n024 | jwc09n027 | jwc09n003 | jwc09n000 | jwc09n024 |

Table 35: AP5 JUWELS Cluster inference benchmark

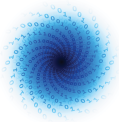
| Experiment number | 1 | 2 | 3 |
|---|---|---|---|
| Data location | /data | /data | /data |
| Job ID | 2166 | 2167 | 2168 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 40 | 40 | 40 |
| Total runtime [s] | 152.02 | 81.42 | 77.36 |
| Model loading [s] | 2.99 | 4.52 | 2.77 |
| Data loading [s] | 110.75 | 60.61 | 59.15 |
| Total inference time [s] | 43.15 | 16.19 | 15.39 |
| Performance [GB/s] | 0.10 | 0.26 | 0.27 |
| Max CPU memory per MPI task [GB] | 31.71 | 31.7 | 31.91 |
| MAX GPU memory per MPI task[GB] | 5.61 | 5.61 | 5.61 |
| Node ID | icnode01 | icnode02 | icnode01 |

Table 36: AP5 E4 Intel System inference runtime.



| Experiment number | 1 | 2 | 3 |
|---|---|---|---|
| Data location | /data | /data | /data |
| Job ID | 2169 | 2170 | 2171 |
| #Nodes | 1 | 1 | 1 |
| #GPUs | 1 | 1 | 1 |
| #MPI Tasks | 1 | 1 | 1 |
| #CPUs per task | 40 | 40 | 40 |
| Total runtime [s] | 68.81 | 70.33 | 68.41 |
| Model loading [s] | 2.44 | 3.96 | 2.01 |
| Data loading [s] | 51.45 | 51.71 | 51.58 |
| Total inference time [s] | 14.83 | 14.60 | 14.82 |
| Performance [GB/s] | 0.28 | 0.29 | 0.28 |
| Max CPU memory per MPI task [GB] | 31.97 | 31.44 | 31.58 |
| MAX GPU memory per MPI task[GB] | 5.27 | 5.27 | 5.27 |
| Node ID | acnode02 | acnode01 | acnode02 |

Table 37: AP5 E4 AMD System inference benchmark



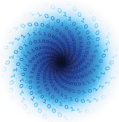
| Experiment number | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Job ID | 7427176 | 7427177 | 7421256 | 7421257 | 7421258 |
| #Nodes | 1 | 1 | 1 | 2 | 4 |
| #GPUs | 1 | 2 | 4 | 4 | 4 |
| #CPUs | 48 | 48 | 48 | 48 | 48 |
| Total runtime [s] | 9,496.58 | 12,658.78 | 15,031.87 | 9,749.03 | 5,278.38 |
| Total training time [s] | 9,496.58 | 12,658.78 | 15,031.87 | 9,749.03 | 5,278.38 |
| Avg. training time per epoch [ms] | 1.84 | 4.59 | 9.72 | 10.24 | 5.34 |
| First epoch training time [ms] | 2.16 | 3.70 | 8.78 | 8.29 | 8.28 |
| Final training loss | 1.94E+00 | 1.85E+00 | 1.95E+00 | 1.95E+00 | 1.94E+00 |
| Node ID | jwb1246 | jwb1247 | jwb1077 | jwb[0985,0991] | jwb[0578,0588,0608,1034] |
| Max. GPU power | 332.95 | 321.57 | 330.04 | 210.03 | 220.54 |
| Avg. GPU power | 68.56 | 72.72 | 86.58 | 81.23 | 81.82 |
| GPU energy consumption [Wh] | 180.86 | 255.71 | 361.52 | 219.98 | 119.97 |

Table 38: AP6 JUWELS Booster training benchmark

| Experiment number | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Job ID | 7427182 | 7427184 | 7421259 | 7421260 | 7421261 |
| #Nodes | 1 | 1 | 1 | 2 | 4 |
| #GPUs | 1 | 2 | 4 | 4 | 4 |
| #CPUs | 48 | 48 | 48 | 48 | 48 |
| Total runtime [s] | 16,266.83 | 17,029.43 | 18,581.30 | 11,343.47 | 5,763.21 |
| Total training time [s] | 16,266.83 | 17,029.43 | 18,581.30 | 11,343.47 | 5,763.21 |
| Avg. training time per epoch [ms] | 3.17 | 6.35 | 12.59 | 13.37 | 6.71 |
| First epoch training time [ms] | 3.01 | 5.17 | 9.65 | 6.76 | 8.59 |
| Final training loss | 1.95E+00 | 1.92E+00 | 1.95E+00 | 1.95E+00 | 1.93E+00 |
| Node ID | jwc09n117 | jwc09n096 | jwc09n183 | jwc09n[069,084] | jwc09n[087,090,093,099] |
| Max. GPU power | 291.17 | 286.18 | 247.71 | 210.03 | 223.31 |
| Avg. GPU power | 55.17 | 62.67 | 76.60 | 71.70 | 69.16 |
| GPU energy consumption [Wh] | 249.29 | 296.45 | 395.37 | 225.92 | 110.72 |

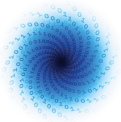
Table 39: AP6 JUWELS Cluster training benchmark

## 6.6 AP 6



| Experiment number | 1 | 2 |
|---|---|---|
| Dataset ID | 2 | 2 |
| Job ID | 2058 | 2140 |
| #Nodes | 1 | 2 |
| #GPUs | 1 | 2 |
| Total runtime [s] | 6,240.740 | 3,936.077 |
| Total training time | 6,240.740 | 3,936.077 |
| Avg. training time per epoch | 2,458.308 | 2,844.843 |
| Final training loss | 1.90E+00 | 1.91E+00 |
| Node ID | icnode01 | icnode[01-02] |
| Avg. Power Consumption [W] | 600.61 | 599.09 |
| Avg. apparent Power [VA] | 620.36 | 617.85 |
| GPU energy consumption [Wh] | 1,041.18 | 1,310.04 |
| Action [MJs] | 23,391.89 | 18,563.09 |

Table 40: AP6 E4 Intel System training benchmark



# Document History

| Version | Author(s) | Date | Changes |
|---|---|---|---|
| | Name (Organisation) | dd/mm/yyyy | |
| 1.0 | E4 | 08/05/2023 | Start internal review |

# Internal Review History

| Internal Reviewers | Date | Comments |
|---|---|---|
| Peter Dueben (ECMWF) | 09/05/2023 | Accepted with minor revisions |

# Estimated Effort Contribution per Partner

| Partner | Effort |
|---|---|
| E4 | 3PM |
| FZJ | 0.5PM |
| **Total** | 3.5PM |